AN ABSTRACT OF THE DISSERTATION OF

Juneki Hong for the degree of Doctor of Philosophy in Computer Science presented on June 7, 2022.

Title: Deep Learning for Human and Biological Languages

Abstract approved: ______________________________

David Hendrix and Liang Huang

We explore the application of deep learning to the disparate fields of natural language processing and computational biology. Both the sentences uttered by humans as well as the RNA and protein sequences found within the cells of their bodies can be considered formal languages in computer science, as sets of strings composed from an alphabet generated by grammar rules. To briefly characterize these languages, words in natural language sentences have a large number of types but a limited sequence of tokens, while nucleotides in biological contexts have limited types in long sequences of tokens. A sentence has a possible vocabulary size greater than 100,000 but in practice usually have less than 20-30 words; RNA sequences have 4 possible tokens but feature sequences anywhere from less than 100 to greater than 10,000 nucleotides. Protein sequences similarly have 20 possible amino acid tokens. The practical differences between these contexts inform our modeling choices to make deep learning tractable and effective, and they further influence what additional algorithms are needed to attain strong results.

These widely different domains presumably have their own forms of syntactic structure, and their respective grammars dictate the relationships on how words, nucleotides, and amino acids interact within themselves to form structures. With language this comes in the form of syntactic parse tree diagrams, with RNA this becomes secondary structure base pairings, and with proteins this becomes tertiary structure contact map pairings. We present a deep learning approach for predicting syntactic structures for human languages (parsing), and dynamic programming techniques that allow for fast linear-time decoding while maintaining close to state-of-the-art accuracy. Converting the traditional $O(n^3)$ exhaustive cubic time CKY parsing algorithm into having a left-to-right, bottom-up reordering allowed us to additionally apply inexact beam search

and then cube-pruning to attain linear $O(n \cdot b \log(b))$ runtime complexity. Despite being an inexact search, our model attained results (91.97 F1) better than the previous state-of-the-art model (91.79 F1) which used an exhaustive decoding upon the same underlying neural network architecture.

Analogous to linguistic grammar rules, nucleotides in RNA sequences are also subject to base pairing potentials, as Adenine (A) prefers to bind with Uracil (U) and Cytosine (C) prefers to bind with Guanine (G). The secondary structure base pairing behavior of RNA often involves interactions across the entire sequence. We present a deep learning approach for predicting secondary structure for RNA sequences (folding), and using self-attention-based Transformer models to visualize and correct errors made by other structure prediction algorithms called bpRNA-Fix. We find that a simple architecture consisting of LSTM and Transformer layers succeed at attaining a strong baseline, which then further improves when predictions made by another program are made available as input. Visualizing the attention weights of our model, we find that strong attention in the last layer is paid towards bracketed structural sections in the output.

We further show a connection to our human language parsing work, by presenting the Nussinov dynamic programming decoding algorithm adapted for deep learning, that guarantees a balanced and valid base pairing output. With cubic runtime complexity analogous to CKY, we show on a dataset of RNA sequences limited to length 50 accuracies surpassing our bpRNA-Fix models. We also discuss how to linearize the runtime which would allow us to scale to longer sequence datasets.

Even more complex than RNA, protein sequences feature even more possible interactions between the 20 different types of amino acids. A typical way to model how a protein sequence will eventually fold into a 3D molecule is to first search for many similar or homologous sequences in a database, and then use the aligned multiple sequence alignment (MSA) as the input, before predicting the distances between each amino acid to every other position, called a contact map. We present a deep learning approach for predicting tertiary structure for protein sequences (contact map prediction), and an algorithm that overall improves the input and output simultaneously by iteratively realigning the former based on the alignment of the latter. Focusing on the cases where little to no homologous sequences can be found for a given input protein sequence (MSA size ≤ 10), we find that the iterative process of realigning the input sequences and output structures results in improvement especially in short, but also in , medium, and long range contacts.

Deep Learning for Human and Biological Languages

by

Juneki Hong

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented June 7, 2022
Commencement June 2022

Doctor of Philosophy dissertation of Juneki Hong presented on June 7, 2022.

APPROVED:

Major Professor, representing Computer Science

Head of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

Juneki Hong, Author

ACKNOWLEDGEMENTS

I would like to thank my advisors Professor David Hendrix and Professor Liang Huang for their advice and guidance throughout my PhD and introducing me to Bioinformatics concepts when I primarily had a different background in Natural Language Processing. They helped me develop as an academic especially in better visualizing and explaining data and models, thinking deeply about algorithms and dynamic programming, and identifying important connections in between different fields.

I would like to thank my committee (In alphabetical order): Professors Alan Fern, Stephen Ramsey, Prasad Tadepalli, and my Graduate Council Representative Professor Jimmy Yang, for all of their questions, suggestions, and advice throughout my different PhD exams, helping me develop sharper presentations and further read into related ideas.

I would like to thank my previous advisors at my past institutions. Professor Roni Rosenfeld at Carnegie Mellon had many words of wisdom for me when I was a masters student and remains one of the nicest people I have ever met. And Professor Jason Eisner at Johns Hopkins, who as incredibly talented as he is still spent a lot of time with me when I was an undergrad, was very supportive of me, and remains a role model that I look up to today.

I would also like to thank both sets of my colleagues at each of the Hendrix and Huang labs, for all the conversations, advice, and support, in addition to ideas, contributions, and collaborations. From the Hendrix lab, this includes Jimmy Bell, Padideh Medisa Danaee, Dezhong Deng, Rosalyn Fey, Patrick Morar, Lillian Padgitt-Cobb, Joseph Valencia, and Nathan Waugh. From the Huang lab, this includes Junkun Chen, James Cross, Ning Dai, Sizhen Li, Kaibo Liu, Mingbo Ma, Matthew Meyn, Göksu Öztürk Miraç, He Zhang, Liang Zhang, Kai Zhao, and Renjie Zheng. I hope to see you all in the future.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

LIST OF ALGORITHMS

To my parents Sungchul and Hyukran Hong, for their love and support.
To my grandparents who always said that I would become a great person and do great things.
To my sweetheart Ayoung Kang who encourages me to do my best every day.

Chapter 1: Introduction

In this thesis, we explore the application of deep learning in the fields of natural language processing (NLP) and bioinformatics, each of which further have traditional roots in the fields of linguistics and biology. We show relations between language and biological contexts by considering these as formal languages or less formally as *sequence problems*; various domain specific issues also arise in our work, and we also discuss these issues. As we relate linguistics and biology[1], we additionally present decoding algorithms, visualization, and explanation methods related to deep learning and used to contribute towards long-standing problems. While in the future machine learning may progress and current deep learning and neural network-based approaches may become obsolete or take on other names, the decoding methods used on top of their outputs and how these models are visualized and interpreted would still remain in relevance.

Both the sentences uttered by humans as well as the RNA and protein sequences found within their cells can be considered as sets of strings composed from an alphabet generated by grammar rules. To briefly characterize these languages, words in natural language sentences have a large number of types but a limited sequence of tokens, while nucleotides in biological contexts have limited types in long sequences of tokens. A sentence has a possible vocabulary size greater than 100,000 but in practice usually have less than 20-30 words[2]; RNA sequences have 4 possible tokens but feature sequences anywhere from less than 100 to greater than 10,000 nucleotides. Protein sequences similarly only have 20 possible amino acid tokens. This leads to researchers in NLP benefitting from encoding words as high dimensional vector embeddings, whereas in biology it typically suffices to encode an RNA nucleotide as a simple categorical *one-hot*. This is one example of practical differences between contexts that inform modeling choices in order to make deep learning tractable and effective.

These widely different domains presumably have their own forms of syntactic structure, and their respective grammars dictate the relationships on how words, nucleotides, and amino

[1]Alan Turing's contributions include solving German cryptography[69], and later in his life he analyzed the random patterns in nature that arise starting from uniformity[68]. Computer scientists have been addressing problems in both language and biology since the beginning of computer science.

[2]A recent study says an average 20 year old recognizes about 42,000 words[3]. There are many more English words than this, Michel et al. 2011 estimates the English lexicon at well over 1 million[44]

acids interact within themselves to form structures. With language this comes in the form of syntactic parse tree diagrams, just like with RNA that this becomes secondary structure base pairings, or for proteins this becomes tertiary structure contact map pairings. We present a deep learning approach for predicting syntactic structures for human languages (parsing), and dynamic programming techniques that allow for fast linear-time decoding while maintaining close to state-of-the-art accuracy. Converting the traditional $O(n^3)$ exhaustive cubic time CKY parsing algorithm into having a left-to-right, bottom-up reordering allowed us to additionally apply inexact beam search and then cube-pruning to attain linear $O(n \cdot b \log(b))$ runtime complexity[3]. Despite being a linear inexact search, our model attained accuracies better than the previous state-of-the-art model which used a slow exhaustive decoding upon the same underlying neural network architecture.

Analogous to linguistic grammar rules, nucleotides in RNA sequences are also subject to base pairing potentials, as Adenine (A) prefers to bind with Uracil (U) and Cytosine (C) prefers to bind with Guanine (G). The secondary structure base pairing behavior of RNA often involves interactions across the entire sequence. We present a deep learning approach for predicting secondary structure for RNA sequences (folding), and using self-attention-based Transformer models to visualize and correct errors made by other structure prediction algorithms called bpRNA-Fix. We find that a simple architecture consisting of LSTM and Transformer layers succeed at attaining a strong baseline, which then further improves when predictions made by another program are made available as input. Visualizing the attention weights of our model, we find that strong attention in the last layer is paid towards bracketed structural sections in the output.

We further show a connection to our human language parsing work, by presenting the Nussinov dynamic programming decoding algorithm adapted for deep learning, that guarantees a balanced and valid base pairing output. With cubic runtime complexity analogous to CKY, we show on a dataset of RNA sequences limited to length 50 accuracies surpassing our bpRNA-Fix models. We also discuss how to linearize the runtime which would allow us to scale to longer sequence datasets.

Even more complex than RNA, protein sequences feature even more possible interactions between the 20 different types of amino acids. A standard way to model how a protein sequence would fold into a 3D molecule is to first search for many similar or homologous sequences in a database, and then use the aligned multiple sequence alignment (MSA) as the input, before pre-

[3]With n being the length of the sequence, and b being the beam size during beam search.

dicting the distances between each amino acid to every other position, called a contact map. We present a deep learning approach for predicting tertiary structure for protein sequences (contact map prediction), and an algorithm that overall improves the input and output simultaneously by iteratively realigning the former based on the alignment of the latter. Focusing on the cases where little to no homologous sequences can be found for a given input protein sequence (MSA size ≤ 10), we find that the iterative process of realigning the input sequences and output structures results in overall improvement.

Chapter 2: Deep Learning for Span-Based Constituency Parsing

2.1 Introduction

Span-based neural constituency parsing [7, 60] has attracted attention due to its high accuracy and extreme simplicity. Compared with other recent neural constituency parsers [15, 41, 14] which use neural networks to model tree structures, the span-based framework is considerably simpler, only using bidirectional RNNs to model the *input sequence* and not the *output tree*. Because of this factorization, the output space is decomposable which enables efficient dynamic programming algorithm such as CKY. But existing span-based parsers suffer from a crucial limitation in terms of search: on the one hand, a greedy span parser [7] is fast (linear-time) but only explores one single path in the exponentially large search space, and on the other hand, a chart-based span parser [60] performs exact search and achieves state-of-the-art accuracy, but in cubic time, which is too slow for longer sentences and for applications that go beyond sentence boundaries such as end-to-end discourse parsing [23, 78] and integrated sentence boundary detection and parsing [2].

We propose to combine the merits of both greedy and chart-based approaches and design a linear-time span-based neural parser that searches over exponentially large space. Following Huang and Sagae (2010) [30], we perform left-to-right dynamic programming in an action-synchronous style, with $(2n-1)$ actions (i.e., steps) for a sentence of n words. While previous non-neural work in this area requires sophisticated features [30, 43] and thus high time complexity such as $O(n^{11})$, our states are as simple as $\ell : (i,j)$ where ℓ is the step index and (i,j) is the span, modeled using bidirectional RNNs without any syntactic features. This gives a running time of $O(n^4)$, with the extra $O(n)$ for step index. We further employ beam search to have a practical runtime of $O(nb^2)$ at the cost of exact search where b is the beam size. However, on the Penn Treebank, most sentences are less than 40 words ($n < 40$), and even with a small beam size of $b = 10$, the *observed* complexity of an $O(nb^2)$ parser is *not* exactly linear in n (see Experiments). To solve this problem, we apply cube pruning [4, 28] to improve the runtime to $O(nb \log b)$ which renders an observed complexity that is linear in n (with minor extra inexactness).

We make the following contributions:

- We design the first neural parser that is both linear time and capable of searching over exponentially large space.[1]
- We are the first to apply cube pruning to incremental parsing, and achieves, for the first time, the complexity of $O(nb \log b)$, i.e., linear in sentence length and (almost) linear in beam size. This leads to an observed complexity strictly linear in sentence length n.
- We devise a novel loss function which penalizes wrong spans that cross gold-tree spans, and employ max-violation update [29] to train this parser with structured SVM and beam search.
- Compared with chart parsing baselines, our parser is substantially faster for long sentences on the Penn Treebank, and orders of magnitude faster for end-to-end discourse parsing. It also achieves the highest F1 score on the Penn Treebank among single model end-to-end systems.
- We devise a new formulation of graph-structured stack [66] which requires no extra book-keeping, proving a new theorem that gives deep insight into GSS.

2.2 Preliminaries

2.2.1 Span-Based Shift-Reduce Parsing

A span-based shift-reduce constituency parser [7] maintains a stack of spans (i, j), and progressively adds a new span each time it takes a shift or reduce action. With (i, j) on top of the stack, the parser can either *shift* to push the next singleton span $(j, j + 1)$ on the stack, or it can *reduce* to combine the top two spans, (k, i) and (i, j), forming the larger span (k, j). After each shift/reduce action, the top-most span is labeled as either a constituent or with a *null* label $\varnothing$, which means that the subsequence is not a subtree in the final decoded parse. Parsing initializes with an empty stack and continues until $(0, n)$ is formed, representing the entire sentence.

[1] `https://github.com/junekihong/beam-span-parser`

$$\begin{array}{ll}
\text{input} & w_0 \dots w_{n-1} \\
\text{state} & \ell : \langle i, j \rangle : (c, v) \\
\text{init} & 0 : \langle 0, 0 \rangle : (0, 0) \\
\text{goal} & 2n-1 : \langle 0, n \rangle : (c, c)
\end{array}$$

$$\textbf{shift} \quad \frac{\ell : \langle \_, j \rangle : (c, \_)}{\ell+1 : \langle j, j+1 \rangle : (c+\xi, \xi)} \; j < n$$

$$\textbf{reduce} \quad \frac{\ell' : \langle k, i \rangle : (c', v') \quad \ell : \langle i, j \rangle : (\_, v)}{\ell+1 : \langle k, j \rangle : (c'+v+\sigma, v'+v+\sigma)}$$

Figure 2.1: Our shift-reduce deductive system. Here ℓ is the step index, c and v are prefix and inside scores. Unlike Huang and Sagae (2010) [30] and Cross and Huang (2016b) [7], ξ and σ are not shift/reduce scores; instead, they are the (best) label scores of the resulting span: $\xi = \max_X s(j, j{+}1, X)$ and $\sigma = \max_X s(k, j, X)$ where X is a nonterminal symbol (could be $\varnothing$). Here $\ell' = \ell - 2(j-i) + 1$.

2.2.2 Bi-LSTM features

To get the feature representation of a span (i, j), we use the output sequence of a bi-directional LSTM [7, 60]. The LSTM produces $\mathbf{f}_0, ..., \mathbf{f}_n$ forwards and $\mathbf{b}_n, ..., \mathbf{b}_0$ backwards outputs, which we concatenate the differences of $(\mathbf{f}_j - \mathbf{f}_i)$ and $(\mathbf{b}_i - \mathbf{b}_j)$ as the representation for span (i, j). This eliminates the need for complex feature engineering, and can be stored for efficient querying during decoding.

2.3 Dynamic Programming

2.3.1 Score Decomposition

Like Stern et al. (2017a) [60], we also decompose the score of a tree t to be the sum of the span scores:

$$
\begin{aligned}
s(t) &= \sum_{(i,j,X)\in t} s(i,j,X) &\quad (2.1)\\
&= \sum_{(i,j)\in t} \max_X s((\mathbf{f}_j - \mathbf{f}_i; \mathbf{b}_i - \mathbf{b}_j), X) &\quad (2.2)
\end{aligned}
$$

Note that X is a nonterminal label, a unary chain (e.g., S-VP), or null label $\varnothing$.[2] In a shift-reduce setting, there are $2n-1$ steps (n shifts and $n-1$ reduces) and after each step we take the best label for the resulting span; therefore there are exactly $2n-1$ such (labeled) spans (i,j,X) in tree t. Also note that the choice of the label for any span (i,j) is only dependent on (i,j) itself (and not depending on any subtree information), thus the max over label X is independent of other spans, which is a nice property of span-based parsing [7, 60].

2.3.2 Graph-Struct. Stack w/o Bookkeeping

We now reformulate this DP parser in the above section as a shift-reduce parser. We maintain a step index ℓ in order to perform action-synchronous beam search (see below). Figure 2.1 shows how to represent a parsing stack using only the top span (i,j). If the top span (i,j) shifts, it produces $(j,j+1)$, but if it reduces, it needs to know the second last span on the stack, (k,i), which is *not* represented in the current state. This problem can be solved by graph-structure stack [66, 30], which maintains, for each state p, a set of predecessor states $\pi(p)$ that p can combine with on the left.

This is the way our actual code works ($\pi(p)$ is implemented as a list of pointers, or "left pointers"), but here for simplicity of presentation we devise a novel but easier-to-understand formulation in Fig. 2.1, where we explicitly represent the set of predecessor states that state $\ell : (i,j)$ can combine with as $\ell' : (k,i)$ where $\ell' = \ell - 2(j-i)+1$, i.e., (i,j) at step ℓ can

[2]The actual code base of Stern et al. (2017a) [61] forces $s(i,j,\varnothing)$ to be 0, which simplifies their CKY parser and slightly improves their parsing accuracy. However, in our incremental parser, this change favors shift over reduce and degrades accuracy, so our parser keeps a learned score for $\varnothing$.

combine with any (k, i) for any k at step ℓ'. The rationale behind this new formulation is the following theorem:

Theorem 1 *The predecessor states* $\pi(\ell : (i, j))$ *are all in the same step* $\ell' = \ell - 2(j - i) + 1$.

Proof. By induction.

This Theorem bring new and deep insights and suggests an alternative implementation that does not require any extra bookkeeping. The time complexity of this algorithm is $O(n^4)$ with the extra $O(n)$ due to step index.[3]

2.3.3 Action-Synchronous Beam Search

The incremental nature of our parser allows us to further lower the runtime complexity at the cost of inexact search. At each time step, we maintain the top b parsing states, pruning off the rest. Thus, a candidate parse that made it to the end of decoding had to survive within the top b at every step. With $O(n)$ parsing actions our time complexity becomes linear in the length of the sentence.

2.3.4 Cube Pruning

However, Theorem 1 suggests that a parsing state p can have up to b predecessor states ("left pointers"), i.e., $|\pi(p)| \leq b$ because $\pi(p)$ are all in the same step, a reduce action can produce up to b subsequent new reduced states. With b items on a beam and $O(n)$ actions to take, this gives us an overall complexity of $O(nb^2)$. Even though b^2 is a constant, even modest values of b can make b^2 dominate the length of the sentence. [4]

To improve this at the cost of additional inexactness, we introduce cube pruning to our beam search, where we put candidate actions into a heap and retrieve the top b states to be considered in the next time-step. We heapify the top b shift-merged states and the top b reduced states. To avoid inserting all b^2 reduced states from the previous beam, we only consider each state's

[3]The word-synchronous alternative does not need the step index ℓ and enjoys a cubic time complexity, being almost identical to CKY. However, beam search becomes very tricky.

[4]The average length of a sentence in the Penn Treebank training set is about 24. Even with a beam size of 10, we already have $b^2 = 100$, which would be a significant factor in our runtime. In practice, each parsing state will rarely have the maximum b left pointers so this ends up being a loose upper-bound. Nevertheless, the beam search should be performed with the input length in mind, or else as b increases we risk losing a linear runtime.

highest scoring left pointer,[5] and whenever we pop a reduced state from the heap, we iterate down its left pointers to insert the next non-duplicate reduced state back into the heap. This process finishes when we pop b items from the heap. The initialization of the heap takes $O(b)$ and popping b items takes $O(b \log b)$, giving us an overall improved runtime of $O(nb \log b)$.

2.4 Training

We use a Structured SVM approach for training [60, 57]. We want the model to score the gold tree t^* higher than any other tree t by at least a margin $\Delta(t, t^*)$:

$$\forall t, s(t^*) - s(t) \geq \Delta(t, t^*).$$

Note that $\Delta(t, t) = 0$ for any t and $\Delta(t, t^*) > 0$ for any $t \neq t^*$. At training time we perform loss-augmented decoding:

$$\hat{t} = \arg\max_t s_\Delta(t) = \arg\max_t s(t) + \Delta(t, t^*).$$

where $s_\Delta(\cdot)$ is the loss-augmented score. If $\hat{t} = t^*$, then all constraints are satisfied (which implies $\arg\max_t s(t) = t^*$), otherwise we perform an update by backpropagating from $s_\Delta(\hat{t}) - s(t^*)$.

2.4.1 Cross-Span Loss

The baseline loss function from Stern et al. (2017a) [60] counts the incorrect labels (i, j, X) in the predicted tree:

$$\Delta_{base}(t, t^*) = \sum_{(i,j,X) \in t} \mathbb{1}\Big(X \neq t^*_{(i,j)}\Big).$$

Note that X can be null $\varnothing$, and $t^*_{(i,j)}$ denotes the gold label for span (i, j), which could also be $\varnothing$.[6] However, there are two cases where $t^*_{(i,j)} = \varnothing$: a subspan (i, j) due to binarization (e.g., a span combining the first two subtrees in a ternary branching node), or an invalid span in t that

[5] If each previous beam is sorted, and if the beam search is conducted by going top-to-bottom, then each state's left pointers will implicitly be kept in sorted order.

[6] Note that the predicted tree t has exactly $2n - 1$ spans but t^* has much fewer spans (only labeled spans without $\varnothing$).

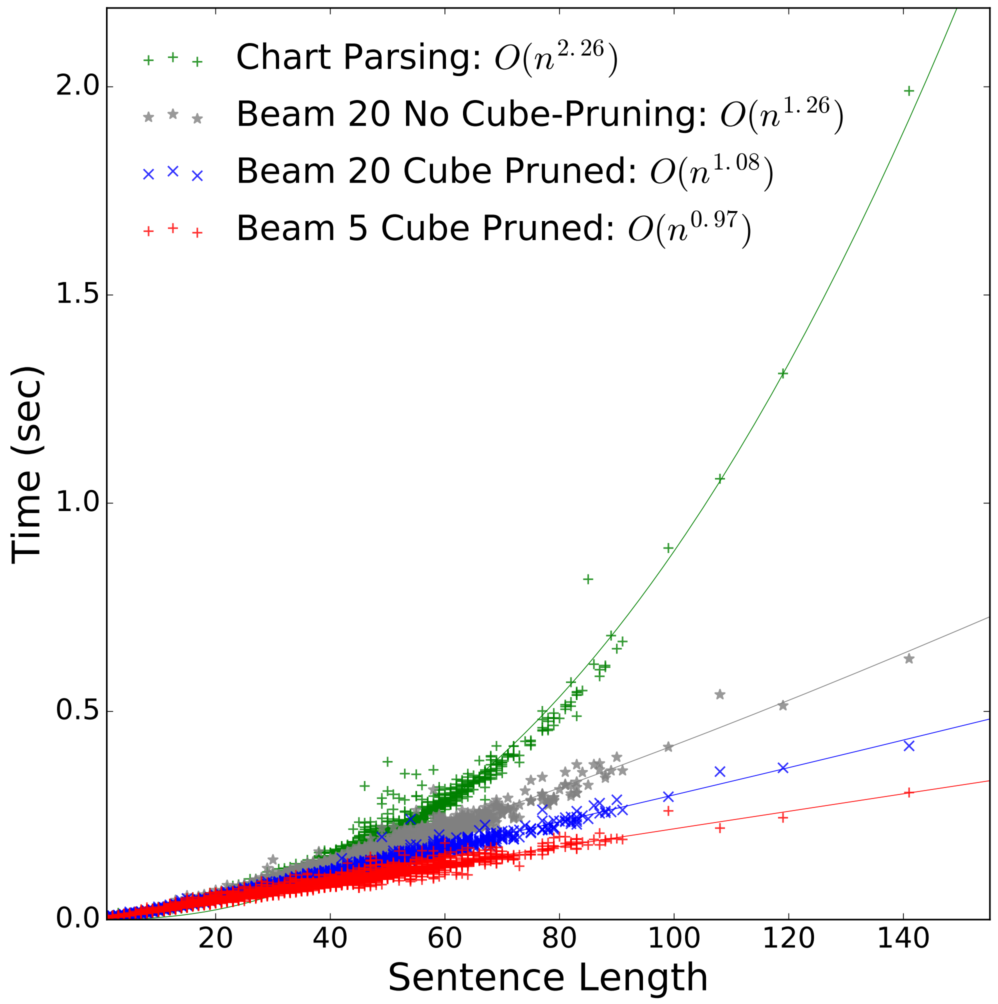


Figure 2.2: Runtime plots of decoding on the training set of the Penn Treebank. The differences between the different algorithms become evident after sentences of length 40. The regression curves have been empirically fitted.

crosses a gold span in t^*. In the baseline function above, these two cases are treated equivalently; for example, a span $(3, 5, \varnothing) \in t$ is not penalized even if there is a gold span $(4, 6, \mathrm{VP}) \in t^*$. So we revise our loss function as:

$$\Delta_{new}(t, t^*) = \sum_{(i,j,X) \in t} \mathbb{1}\Big(X \neq t^*_{(i,j)} \vee \text{cross}(i, j, t^*)\Big)$$

where $\text{cross}(i, j, t^*) = \exists\ (k, l) \in t^*$, and $i < k < j < l$ or $k < i < l < j$.

2.4.2 Max Violation Updates

Given that we maintain loss-augmented scores even for partial trees, we can perform a training update on a given example sentence by choosing to take the loss where it is the greatest along the parse trajectory. At each parsing time-step ℓ, the *violation* is the difference between the highest augmented-scoring parse trajectory up to that point and the gold trajectory [29, 75]. Note that computing the violation gives us the max-margin loss described above. Taking the largest violation from all time-steps gives us the max-violation loss.

2.5 Experiments

We present experiments on the Penn Treebank [42] and the PTB-RST discourse treebank [78]. In both cases, the training set is shuffled before each epoch, and dropout [24] is employed with probability 0.4 to the recurrent outputs for regularization. Updates with minibatches of size 10 and 1 are used for PTB and the PTB-RST respectively. We use Adam [35] with default settings to schedule learning rates for all the weights. To address unknown words during training, we adopt the strategy described by Kiperwasser and Goldberg [37]; words in the training set are replaced with the unknown word symbol UNK with probability $p_{unk} = \frac{1}{1+f(w)}$, with $f(w)$ being the number of occurrences of word w in the training corpus. Our system is implemented in Python using the DyNet neural network library [45].

2.5.1 Penn Treebank

We use the Wall Street Journal portion of the Penn Treebank, with the standard split of sections 2-21 for training, 22 for development, and 23 for testing. Tags are provided using the Stanford tagger with 10-way jackknifing.

Table 2.1 shows our development results and overall speeds, while Table 2.2 compares our

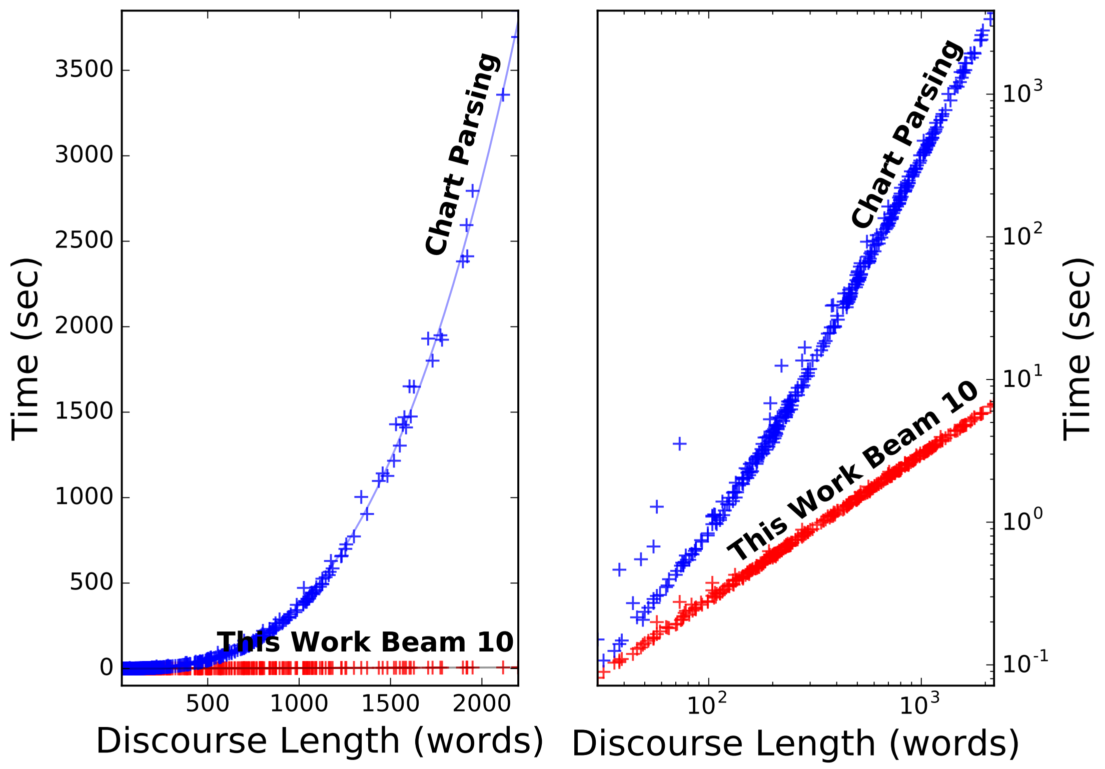


Figure 2.3: Runtime plot of decoding the discourse treebank training set. The log-log plot on the right shows the cubic complexity of baseline chart parsing. Whereas beam search decoding maintains linear time even for sequences of thousands of words.

test results. We show that a beam size of 20 can be fast while still achieving state-of-the-art performances.

2.5.2 Discourse Parsing

To measure the tractability of parsing on longer sequences, we also consider experiments on the PTB-RST discourse Treebank, a joint discourse and constituency dataset with a combined representation, allowing for parsing at either level [78]. We compare our runtimes out-of-the-box in Figure 2.3. Without any pre-processing, and by treating discourse examples as constituency trees with thousands of words, our trained models represent end-to-end discourse parsing systems.

For our overall constituency results in Table 2.3, and for discourse results in Table 2.4, we

	Development Set 22 (F1)		
	Baseline	+cross-span	Time
This Work Beam 1	89.41	89.93	0.042
This Work Beam 5	91.27	91.91	0.050
This Work Beam 10	91.56	92.09	0.058
This Work Beam 15	91.74	92.16	0.062
This Work Beam 20	91.65	92.20	0.066
Chart	92.02	92.21	0.076

Table 2.1: Comparison of PTB development set results, with the time measured in seconds-per-sentence. The baseline chart parser is from Stern et al.[61], with null-label scores unconstrained to be nonzero, replicating their paper.

	Test Set 23		
End-to-End & Single Model	LR	LP	F1
Socher et al. (2013) [59]			90.4
Durrett and Klein (2015) [14]			91.1
Cross and Huang (2016) [7]	90.5	92.1	91.3
Liu and Zhang (2016) [41]	91.3	92.1	91.7
Dyer et al. (2016) (discrim.) [15]			91.7
Stern et al. (2017a) [60]	90.63	92.98	91.79
Stern et al. (2017a) +cross-span [60]	91.67	91.94	91.81
Stern et al. (2017b) [61]	91.35	92.38	91.86
This Work Beam 10	91.44	91.91	91.67
This Work Beam 15	91.64	92.04	91.84
This Work Beam 20	91.49	92.45	**91.97**
Reranking/Ensemble/Separate Decoding/Aug Data			
Vinyals et al. (2015) (ensem) [71]			90.5
Dyer et al. (2016) (gen., rerank) [15]			93.3
Choe and Charniak (2016) (rerank) [5]			93.8
Stern et al. (2017c) (sep. decoding) [62]	92.57	92.56	92.56
Fried et al. (2017) (ensem/rerank) [18]			94.25

Table 2.2: Final PTB Test Results. We compare our models with other (neural) single-model end-to-end trained systems, trained on WSJ only.

	LR	LP	F1
Zhao and Huang (2017) [78]	81.6	83.5	82.5
This Work Beam 10	80.47	80.61	80.54
This Work Beam 20	80.86	80.73	80.79
This Work Beam 200	81.51	80.84	81.18
This Work Beam 500*	81.50	80.81	81.16
This Work Beam 1000*	81.55	80.85	81.20

Table 2.3: Overall test accuracies for PTB-RST discourse treebank. Starred* rows indicate a run that was decoded from the beam 200 model.

	segment	structure	+nuclearity	+relation
Bach et al (2012) [1]	95.1	-	-	-
Hernault et al. (2010) [23]	94.0	72.3	59.1	47.3
Zhao and Huang (2017) [78]	95.4	78.8	65.0	52.2
This Work Beam 200	91.20	73.36	58.87	46.38
This Work Beam 500*	93.52	74.93	60.16	47.03
This Work Beam 1000*	94.06	75.60	60.61	47.37

Table 2.4: F1 scores comparing discourse systems. Results correspond to the accuracies in Table 2.3, broken down to focus on the discourse labels.

adapt the split-point feature described in [78] in addition to the base parser. We find that larger beamsizes are required to achieve good discourse scores.

2.6 Conclusions

We have developed a new neural parser that maintains linear time, while still searching over an exponentially large space. We also use cube pruning to further improve the runtime to $O(nb \log b)$. For training, we introduce a new loss function, and achieve state-of-the-art results among single-model end-to-end systems.

Chapter 3: Deep Learning for Fixing RNA Secondary Structure Prediction

3.1 Introduction: bpRNA-Fix

Ribonucleic acids (RNA) are a ubiquitous and essential molecule for life, and the structures of RNA molecules are widely understood to inform their function. Much research is devoted to the fascinating aspect of their structure preserving properties, in that RNA sequences can vary widely in a myriad of different examples but can still all fold into similar structures. The task of predicting the secondary structure of RNA sequences were traditionally approached with thermodynamics-based dynamic programming algorithms[25] and machine learning derived parameters[12, 31]. However, recent advances in the application of deep learning have demonstrated progress in RNA structure prediction performances in accuracy while still maintaining tractability[19, 67, 56].

More broadly, there has been interest in utilizing recent advanced deep learning models, such as using the attention-based Transformer model[70], which was original designed for translation between languages. This network has been applied to a wide variety of tasks within biological domains including protein structure[32], drug-target interaction[27], DNA enhancer prediction[38], and DNA sequence classification[22]. Transformers utilize self-attention, which describe relationships between elements of the input, and can be integrated to compute the elements of an output. This combination of information is not necessarily uniform in distribution, and is instead weighted differently for each part of the input sequence depending its importance deemed by the neural network. Thus the term “self-attention” comes from representing a element from the input sequence in relation to all elements including itself and the neural network’s model weights.

While thermodynamic models of secondary structure prediction have readily interpretable parameters, deep learning models generally require additional tools for interpretation [58, 47, 77]. Attention weights when applied to RNA secondary structure suggest an attractive means of network interpretation because one can visualize the relevant parts of the input nucleotide sequence that inform the network’s decision on forming a base pair. However, a notable theoretical limitation of self-attention based neural networks is the inability to pair an arbitrary number of brackets[21], an important property common in biological sequence related tasks, including the prediction of base pairing. In fact, common representations of RNA secondary structures include "dot bracket" sequences, which represent base pairs as a pair of matched parentheses.

To partially alleviate a part of what Transformers may find difficult about RNA folding, we use the output of the thermodynamic models as part of the input, which already have matching base pair brackets. Allowing this additional input to form a "scaffolding", we aid the network in providing an already well-formed base pair bracketing such that during training its learning efforts would focus on other aspects in the secondary structure prediction (RNA folding) task. We call this task *base pair fixing*, where a previous model's predictions (e.g. using the output of Linearfold or RNAfold) are input in addition to the nucleotide sequence and the output dotbracket positions that differ from the input are considered *fixes* to the input.

Base pair fixing offers an additional way towards neural network explanation and visualization, compared to sequence-only prediction approaches. That is, self-attention weights have an inherent directionality to them, and one can specifically focus on and visualize the attention to positions that are corrected in the output, and even categorized across different types of fixes to be analyzed. We explore whether our model's attention weights are directly interpretable in this way.

A machine learning advantage of this approach is that the usage of another structure prediction program indirectly makes use of their previously designed thermodynamic models, standing on their proverbial shoulders in order to simplify our task and expedite training. In addition, the input models were designed for more general RNA folding task use cases. Whereas almost any given RNA dataset may have biases towards certain particular types or families or RNAs, away from what the input programs were originally designed for.

Thus, an alternate perspective on base pair fixing considers our models to be trained to fine-tune upon fixing the "errors" that Linearfold or RNAfold may make on our dataset. Fine-tuning from a previous model is by no means a novel approach and is already a widespread method in deep learning related tasks, such as in tasks related to Natural Language Processing and Computer Vision leveraging the BERT[9] and Resnet[64] pretrained deep learning models respectively. Where these fine-tuning methods rely on very large deep learning models trained an extremely large datasets and having greatly in excess of millions of parameters, our approach utilizes much smaller and faster to compute base programs. We show that the existence of a cheaper or simpler model to query can still effectively serve to reduce time and effort needed to train a neural network. For parity tasks such as RNA secondary structure prediction, the already basepair-matched input obviates part of the burden of learning to maintain balanced dotbracket pairing. This idea can be applied beyond RNA Secondary Structure Prediction or even in Biological problem domains; information from another possibly “simpler” model can still be incorporated or ensembled for higher accuracy performances. We also compare a version of our model relying only on the underlying neural network model without additional input predictions, that we call “AttentionFold”.

3.2 Methods

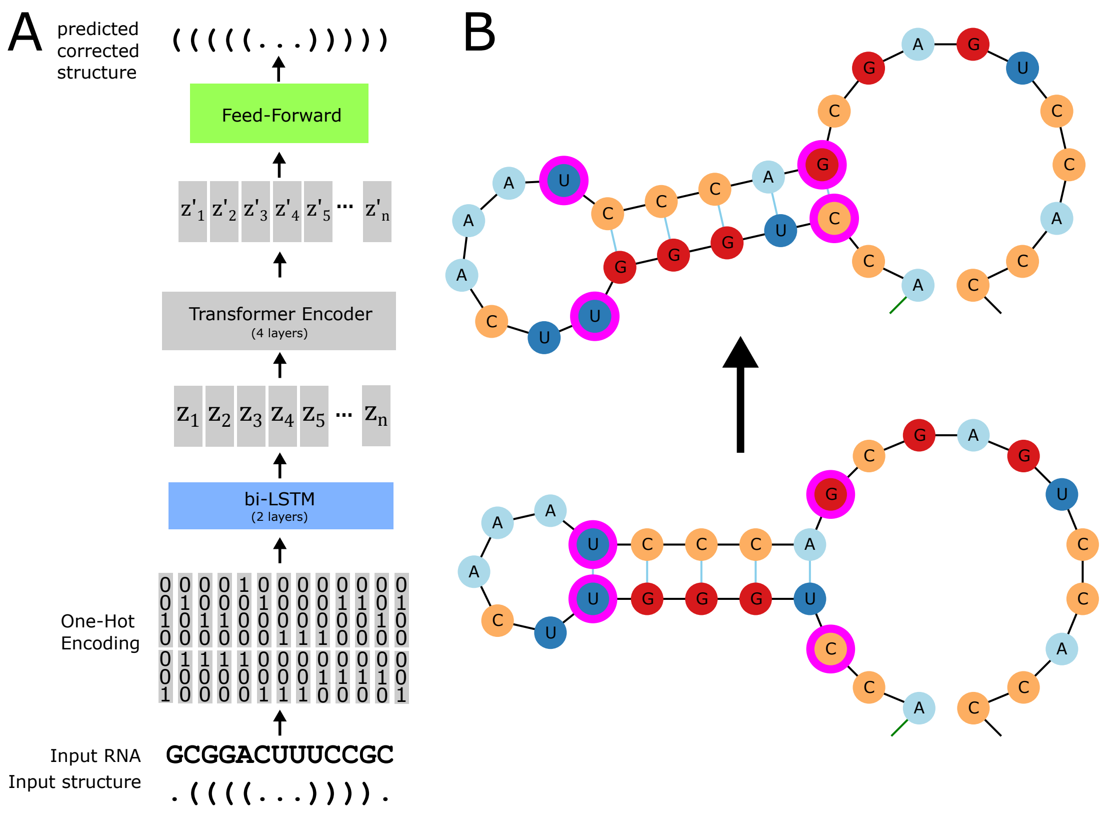


Figure 3.1: A: Architecture Diagram for the Basepair Fixing task. B: A simple example of an input RNA sequence (bottom) becoming “fixed” by our model (top), with the fixed positions highlighted in magenta. In this case the added base-pairs shift the pairings of the entire stem.

3.2.1 Transformer Networks

Transformer[70] Networks are a Deep Learning model that utilize self-attention to model sequential data[9] or otherwise structured data, such as images or graphs[48, 76]. In biological sequential data such as RNA, each nucleotide is represented by a rich multi-dimensional vector, and to procedurally apply a "layer" of a Transformer Network would be to update each nucleotide vector representation using a series of matrix multiplications involving all nucleotides in the sequence.

More formally, x_i is the d-sized vector embedding of the nucleotide at position i, and x represents the entire RNA sequence; an instance of the Transformer model learns three weight matrices W_Q (query weights), W_K (key weights), and W_V (value weights), then the resulting matrices:

$$Q = xW_Q, K = xW_K, V = xW_V$$

Are combined together, divided by $\sqrt{d_k}$ the square root of the dimension-size of W_K, and utilizing the softmax function, in the following way:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

The resulting output amounts to updating every x_i to an enriched version x'_i, which incorporates information from every position. These attention units are typically stacked in multiple "layers", and the final sequence representation is utilized for a downstream task, for example such as site-wide classifications of whether each nucleotide is paired or unpaired, or sequence-wide predictions such as for an existence of a pseudoknot. Transformer Networks additionally have *encoder* and *decoder* components, that both utilize the self-attention mechanism detailed above. For our purposes, we utilize only encoder layers, as our decoder is a simple feed-forward network to make site-wide predictions.

3.2.2 Architecture and Training

Reading in the sequence and input structure represented as a one-hot encoding, the input is richly encoded with the use of a two-layer bi-LSTM, followed by four layers of Transformer self-attention encoders. The final output is then predicted with a simple feed forward network, such that each position of the sequence independently has three possible outputs (left-paired, right-paired, unpaired). This description is summarized in Figure 3.1.

To note, this decoding scheme is unconstrained, in that it is entirely possible to output nonsensical, mismatching, or unbalanced base pairs; the model is relied upon to output matching base pairs on its own without a formal guarantee that will only produce perfectly valid outputs. A myriad of solutions could exist to address this, from simple post-processing of the output to ensure basepair balance[1], to running a compatible version of the Nussinov algorithm on the output embeddings[2] to find the highest model-scoring valid basepairing, we instead limit the scope of this paper to a simple decoding method for pedagogical reasons and to explore and discuss the model's explainability.

Our Deep Learning model and training was coded using the Pytorch library, with a Cross-Entropy training loss and the Adam[36] optimizer[3]. Since the network can predict every position of a sequence as well as every sequence in a minibatch in parallel, the training loss can all be computed with vector notation within a GPU for efficiency.

3.2.3 Dataset and Preparation

Starting with the sequences in the bpRNA_1M dataset[8], sequences with lengths between 20 and 200 were kept. The remaining sequences were further filtered for diversity using CD HIT-EST[40]. This data was split randomly using an 80-10-10 training, validation, and testing split. This consideration addresses over-fitting, for the potential issue of training examples to be very similar to those found in the validation and test, as well as an indirect way to address separating RNA sequence types and families within the data.

As a way to increase the number of training examples, some of the sequences excluded during the diversity filtering were added back into the training data. Using the EMBOSS Needleman-Wunsch global alignment program, the selected sequences had to have had at least 90% similarity

[1]For example, adding or deleting mismatched outside basepairs until the overall structure is balanced. Or reading the structure from left to right using a stack, pushing and popping the stack based on whether left or right brackets are read, and then deleting any right-brackets that would cause the stack to be popped while empty or any remaining left-brackets that remain on the stack by the end of the sequence.

[2]This would require a scoring function that takes two embeddings representing nucleotides and returns a score for how well they would pair. This could easily be a simple feedforward network either learned jointly while training the rest of the network (which would offer the best final results but could be very slow to train), or by holding the weights of the rest of the network frozen after it has already been trained using our baseline decoding (as a machine learning engineering compromise).

[3]The Adam hyperparameter were all default values from the Pytorch library other than the learning rate (0.001), betas values (0.9, 0.98), and epsilon value (1e-09).

Symbol	(	)	.	Label
(	46521	461	5553	52535
)	385	46616	5534	52535
.	4371	4304	125706	134381
Predicted	51277	51381	136793	239451
Precision	90.7%	90.7%	91.9%	91.4%

Table 3.1: Confusion matrix summarizing the performative accuracy of our Linearfold+bpRNAfix model on our held-out test set, with our overall site-wise accuracy shown in the bottom right-hand corner.

with at least one of the training sequences but with none of the validation and testing sequences. The final resulting process resulted in a split consisting of 42,859, 2,344, and 2,345 training, validation, and testing sequence examples respectively.

3.3 Results

3.3.1 Positional Encodings vs LSTM

There is also a notable design shortcoming in Transformer network architectures in that attentions are computed without natural notions of position. Thus all x'_i resulting output embeddings incorporate information from inputs x_j from all positions j without even considering whether $i < j$ for example. This is typically addressed with the addition of a positional encoding, usually a trigonometric function with a long period, to encode every position index. The positional encoding is then combined with the input embedding, allowing the model to differentiate between nucleotides occurring at different positions.

Instead of using a positional encoding, we instead turn to another common neural network architecture in the Bidirectional Long-Short Term Memory (bi-LSTM), designed to model sequential information. Using an LSTM as the first encoding layer of our network, we forgo the usage of Positional Encodings, and instead allow the network to model relative sequential information as a part of its learning. We find that this leads to an overall improvement in final evaluation performance in Figure 2A.

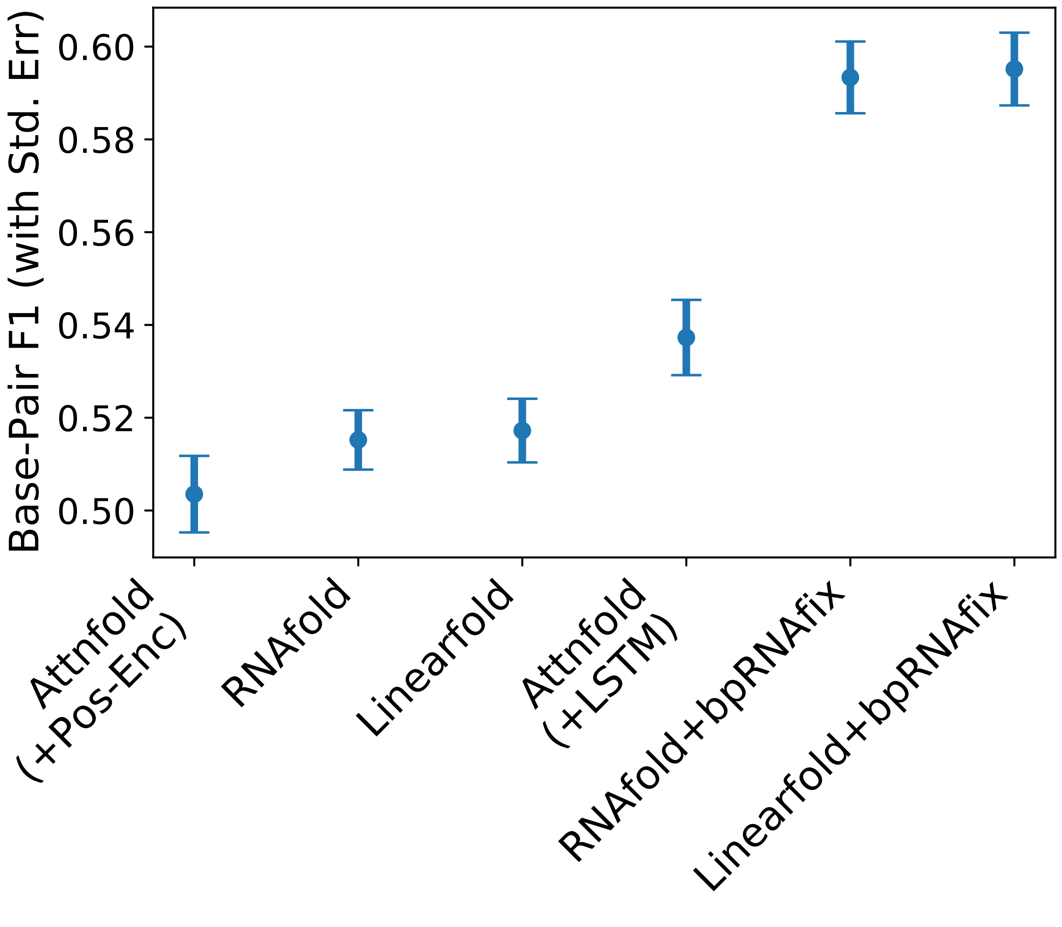


Figure 3.2: Sequence-level F1 score performances with standard error bars for each of our models evaluated on the test set.

Model	Precision	Recall	F1	Seq Avg F1
RNAfold	45.97	**62.50**	52.97	51.52
Linearfold	54.96	57.68	**56.29**	51.72
Attentionfold (+Pos Enc)	47.32	45.64	46.47	50.35
Attentionfold (+LSTM)	49.92	48.57	49.24	53.73
bpRNA-Fix (+RNAfold)	57.16	55.11	56.12	59.34
bpRNA-Fix (+Linearfold)	**57.58**	54.96	**56.24**	**59.52**

Table 3.2: Evaluated test results for our models. Precision, recall, and F1 are calculated across all base pairs in the test set. Sequence average test F1 refers to the average F1 score across all evaluated sequences. Figure 3.2 plots the sequence average F1 with standard error bars.

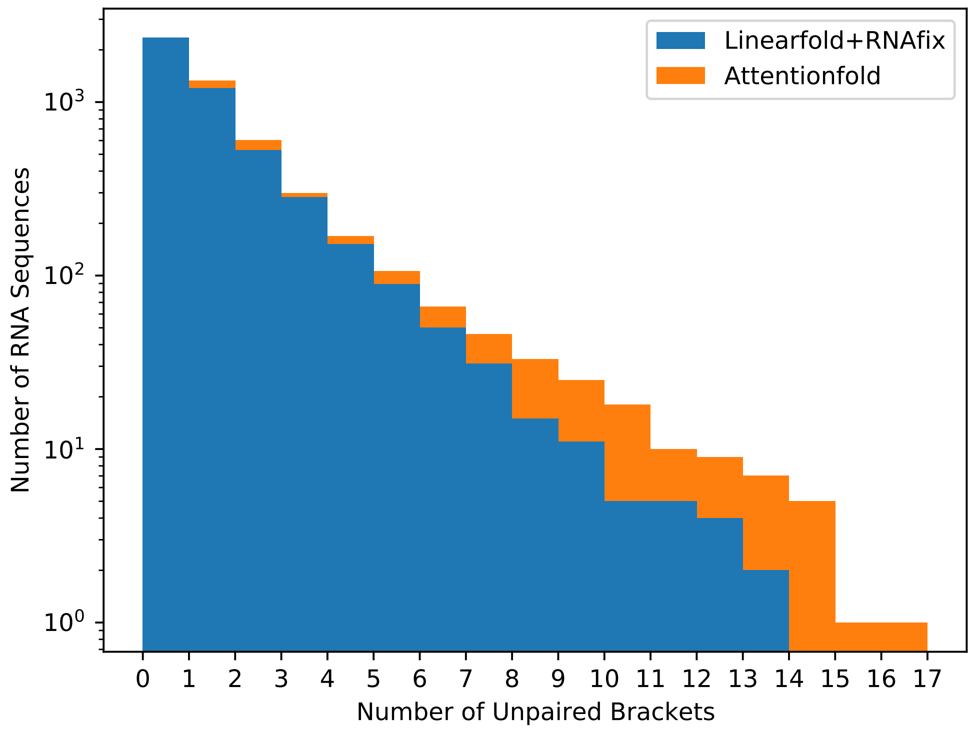


Figure 3.3: Reverse-CDF of the number of unpaired brackets produced by bpRNA-Fix vs Attentionfold on the test dataset. Further quantifying the performance difference between the two models, Attentionfold tends to produce more unpaired brackets in its output than bpRNA-Fix.

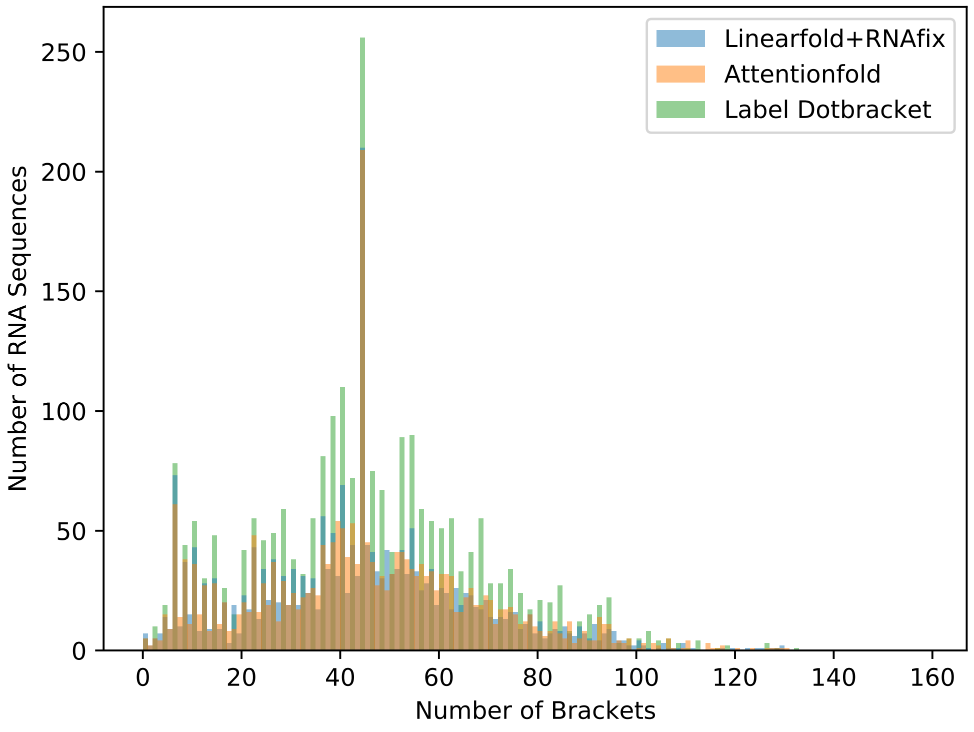


Figure 3.4: Histogram binning the number of brackets predicted by our models compared to the labeled dataset. bpRNA-Fix tends to predict more even and less odd numbers of basepairs compared to Attentionfold.

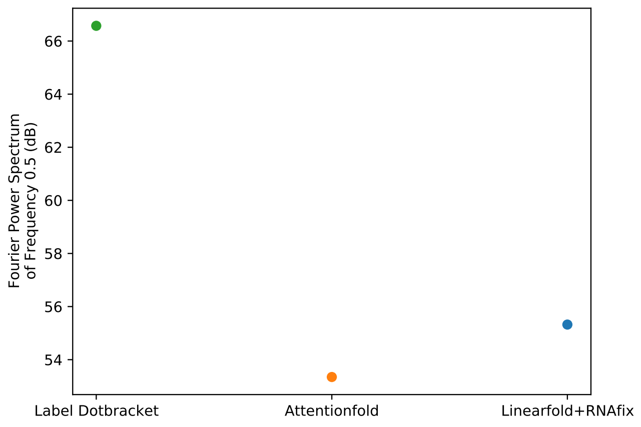


Figure 3.5: Treating Figure 3.4 histogram as time-series data, we apply a Fast-Fourier Transformation to measure the parity of the model output predictions. The resulting peaks at frequency 0.5 (corresponding to an even parity of base pair predictions) are compared in this plot, showing that bpRNA-Fix predicts base pairs with higher even parity than Attentionfold

3.3.2 F1-Score Evaluation of Secondary Structure

In order to further compare the performances of RNAfold and Linearfold to our bpRNA-Fix methods, the first-pass approach for evaluation would be to calculate the site-wise confusion matrix and accuracy. Indeed, Table 3.1 shows a robust overall accuracy greater than 91%. However, due to the unconstrained nature of our dotbracket predictions, it is still possible to do well on this metric while still having inconsistent on nonsensical base pairings.

In order to more directly take RNA secondary structure base pairings into account, we use an evaluation metric based on the precision and recall of the (i, j) base pairs in the labeled structure. To compute the basepairings, we use a simple program that reads a dotbracket from left to right using a stack data structure. Left brackets are pushed on to the stack and right brackets pop from the stack, forming a series of (i, j) pairs as a result. In the case of invalid or mismatched brackets, right brackets that try to pop from an empty stack are ignored as well as remaining left brackets on the stack after the entire dotbracket has been processed. In this way, we can extract the corresponding series of base pairs from the predicted output even if the prediction is unbalanced, and use this to compare with the labeled structure.

For each test sequence we calculate the resulting F1-Score, which is the harmonic mean of

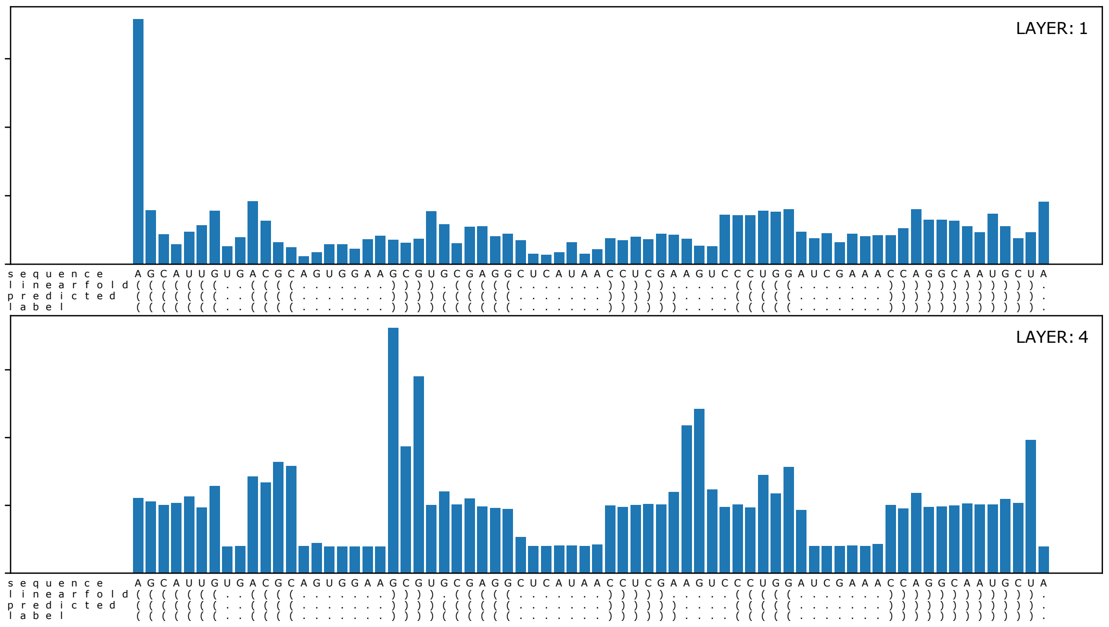


Figure 3.6: Histogram of bpRNA-Fix's attention weights for a specific bpRNA sequence (bpRNA_CRW_43036). The summed attention weights outgoing from each position are binned. The first and last layers are shown. While the initial attention weights in layer one look to be more uniform in nature, the weights become more organized by the fourth layer, concentrating around regions with structure.

the precision and recall, and the average and standard error of this evaluation what we show in Figure 3.2. This evaluation is pessimistic in that a single missed or added base pair can mismatch an entire set of index pairings, and thus potentially ruin the subsequent evaluated F1-Score; it's especially pessimistic for our unconstrained model because we predict each position independently potentially making invalid or unbalanced bracket outputs possible. Even though this potentially advantages the always balanced outputs of RNAfold and Linearfold we still show a surprisingly strong performance in that the trained Attentionfold baseline model with LSTM+Transformer network layers still outperforms the thermodynamic models, and then improves even further as the RNAFix model upon incorporating the thermodynamic model outputs. Linearfold and RNAfold were not designed to fold the sequences in the bpRNA_1M dataset, but these results still show that deep learning based approaches can improve upon off-the-shelf structure prediction programs.

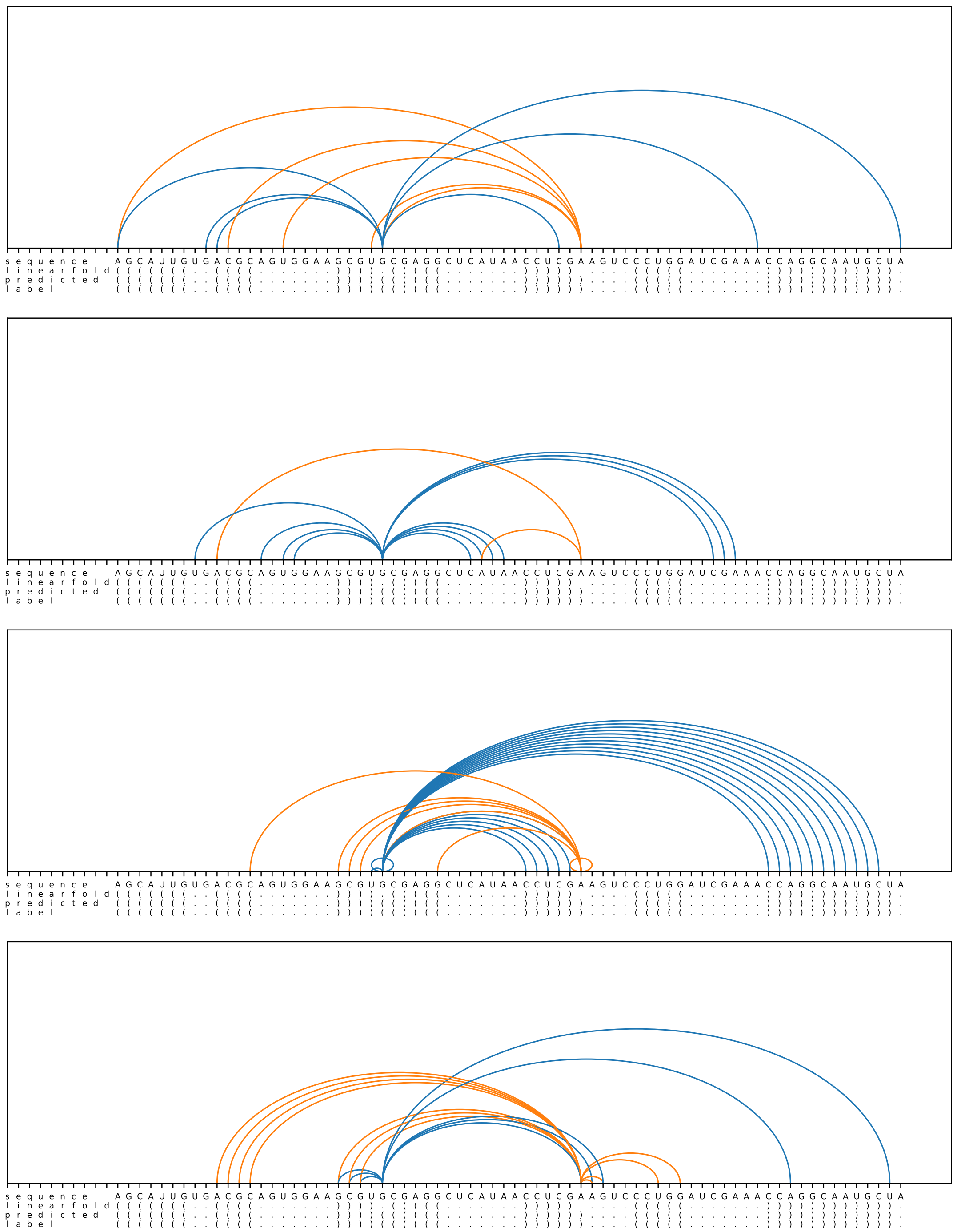


Figure 3.7: Arc plot of the same sequence as Figure 3.6, showing the statistically significant attention values to fixed positions, of the first and last self-attention layers of the model. The resulting arc plot shows that significant attentions are coming from structured regions in this example, and that the significant attentions behave differently from layer to layer.

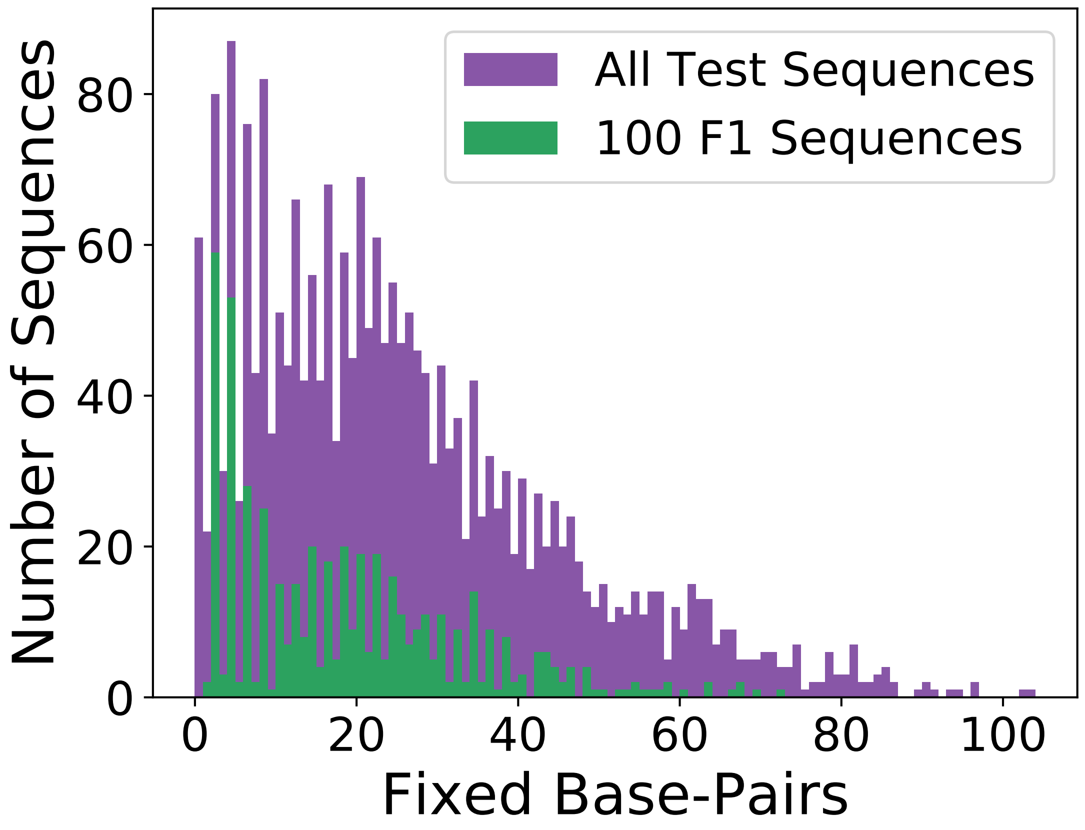


Figure 3.8: Number of sequences of the test dataset binned by the number of fixed base-pairs. We show an even parity, that many sequences tend to have an even number of fixes, even though the model is not constrained to do so. For comparison, we also show the subset of test sequences where our model perfectly predicted the dotbracket structure.

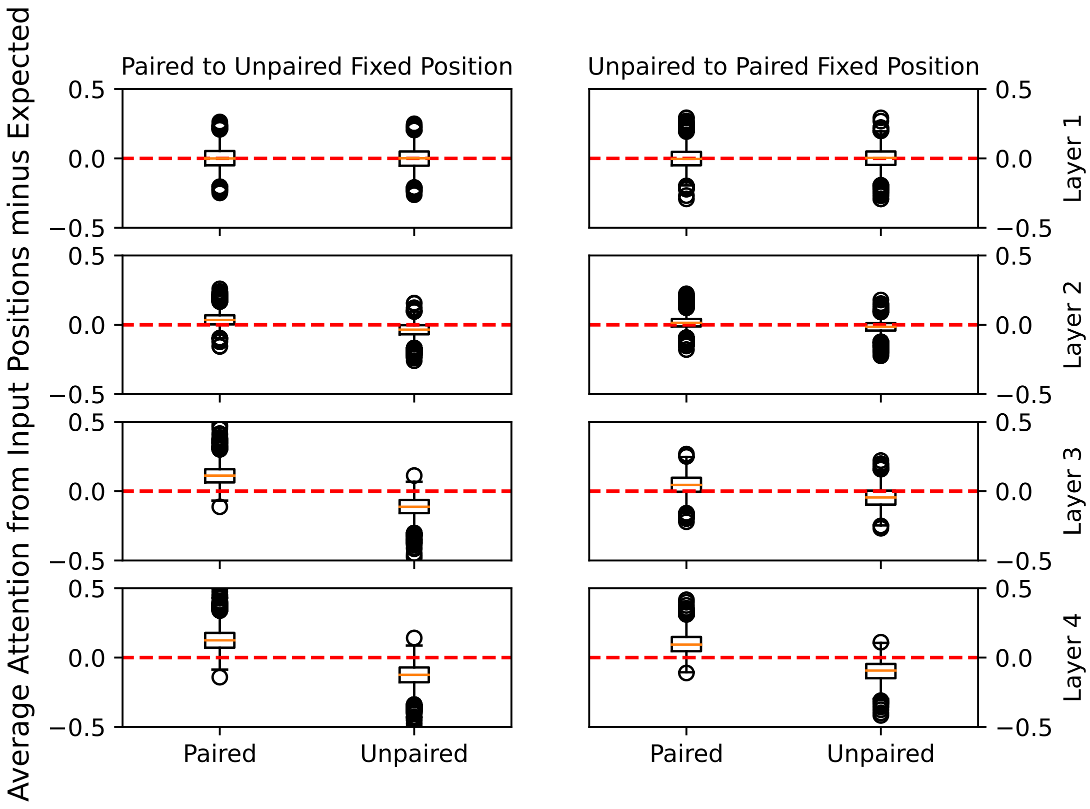


Figure 3.9: Boxplots showing attentions to fixed positions across the test dataset, subdivided by whether the output fixed position is a paired or unpaired fix (left or right column) and whether the attentions are coming from paired or unpaired input positions (left or right boxplot). Only sequences that our model get correct (100 F1) are considered. We see that both paired and unpaired fixes, attention weights from paired positions across the input are higher than a uniform expected baseline, especially in subsequent layers of the network.

3.4 Model Interpretation

In addition to better folding performances, the task of bpRNA fixing has an added benefit of having additional avenues for model interpretation via introspection of attention weights at positions where the thermodynamic inputs differ from the predicted outputs (fixed positions).

Focusing on fixed positions in the output, we show that attention weights are stronger coming from paired regions of the input compared to a uniform baseline, as seen in Figure 3.9. Looking at particular predicted sequence examples, such as in Figure 3.6, where the summed attention weights emanating from each position as well as Figure 3.7 where strong individual attentions going towards a fixed position are visualized, we see overall stronger attention weights coming from input paired positions.

3.4.1 Summed Attention Weights

Attention weights in neural networks are typically normalized, allowing for a distributional interpretation across the input sequence. Thus, attention weights from all positions towards a particular position sums to one. A first approach towards attention weight interpretation simply reversing this summation, binning the attention weights emanating *from* each position. An example of this can be seen in Figure 3.6 where we see the summed attention weights become more organized around paired regions in subsequent layers of the network.

3.4.2 Individual Attention Weight Visualization

To start delving into the summed attention weights to find *which* individual attention weights are more significant than others, a notion of statistical significance via a Z-Score[4] can be devised from the distribution of attentions from all other positions. Looking at the attentions to a fixed position, we can show results such as in Figure 3.7, where only arcs that have Z-Score greater than one are considered.

[4]Z-Score is computed by taking the mean μ and standard deviation σ from all the attention weights x_i across all positions i. The Z-Score of a particular attention weight is thus $z = \frac{x-\mu}{\sigma}$

3.4.3 Differentiation of network layers

The summed attention weights and individual arc visualizations show evidence that the layers in our self-attention model tend to specialize or differentiate in roles. These individual cases are also substantiated as overall trends in Figure 3.10, where the distributional nature of the attention weights are quantified and then plotted layer-by-layer as a heatmap, showing that indeed subsequent layers distributionally behave differently than previous ones. Neural Networks have been shown to specialize by layer in many different problem domains[50, 49], and we also demonstrate this property in our model as well.

3.4.4 Non-Uniformity: Information Content

Exploring individually strong attention weights leads to a discussion of non-uniformity in that attention weights to a position with one or more significant individual weights should tend away from a uniform distribution. We quantify this notion of non-uniformity by utilizing information content. Summarized in Figure 3.10, the information Content heatmap measured at each layer of our Transformer attention network shows differentiation across layers in fix cases, as well as strong peaked attentions when fixing left brackets into unpaired, while not nearly as much as fixing the corresponding right brackets into unpaired. This suggests that the decision-making process involving removing base pairs is not symmetrical over the base pair itself.

We define information content as the following. For an $n \times n$ weight matrix W, for $\forall i, j \in [0, n]$ the weight $w_{ij} \in W$ indicates the weight of the attention from indices j to i. The attention weights to a given index are softmax normalized from the Transformer Network, thus $\sum_j w_{ij} = 1$.

information content is calculated in terms of a baseline (expected) entropy minus an entropy calculated from an observed distribution. So the information content of a given index i is:

$$IC_i = \bar{H}_i - \hat{H}_i$$

If we invoke the commonly used uniform baseline, the expected entropy at that index becomes:

$$\bar{H}_i = -\sum_{j=0}^{n} \left(\frac{1}{n} \cdot \log_2 \frac{1}{n} \right) = \log_2 n$$

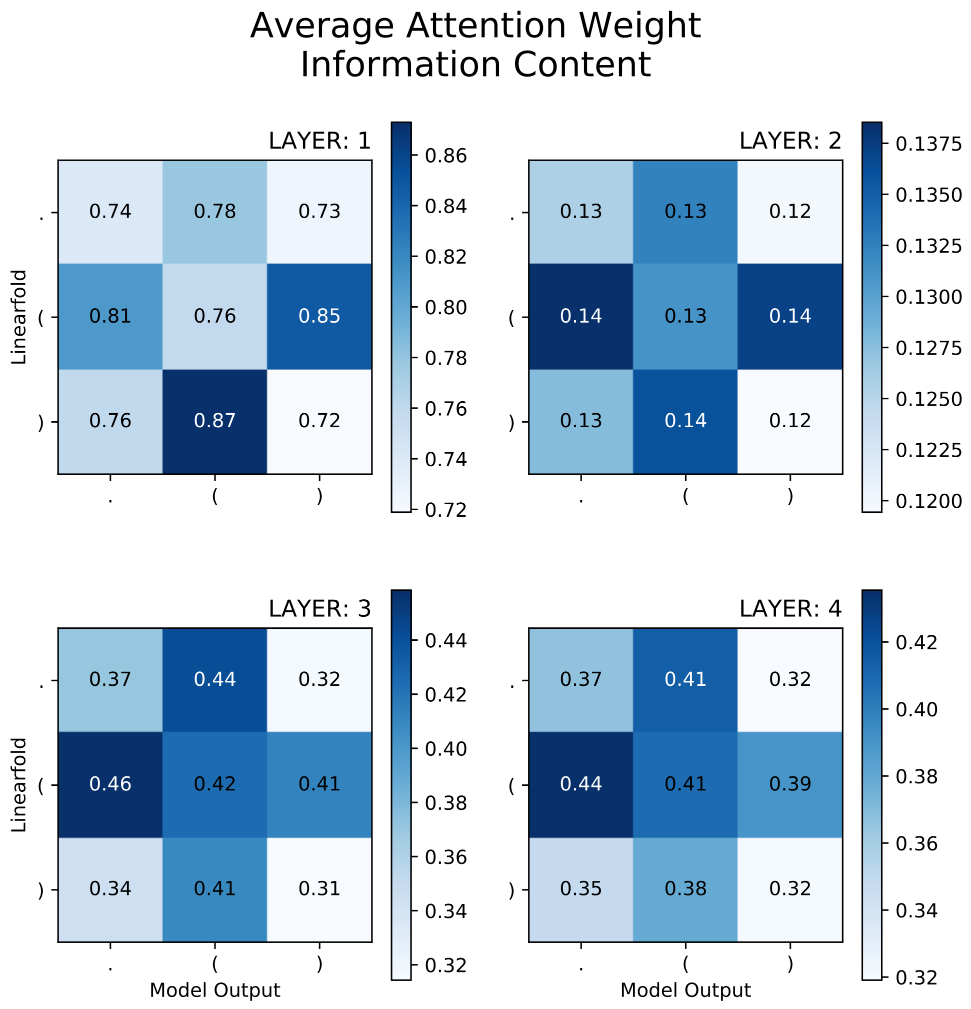


Figure 3.10: Average incoming attention weight information content, for each modification from Linearfold made by our model in the test data set.

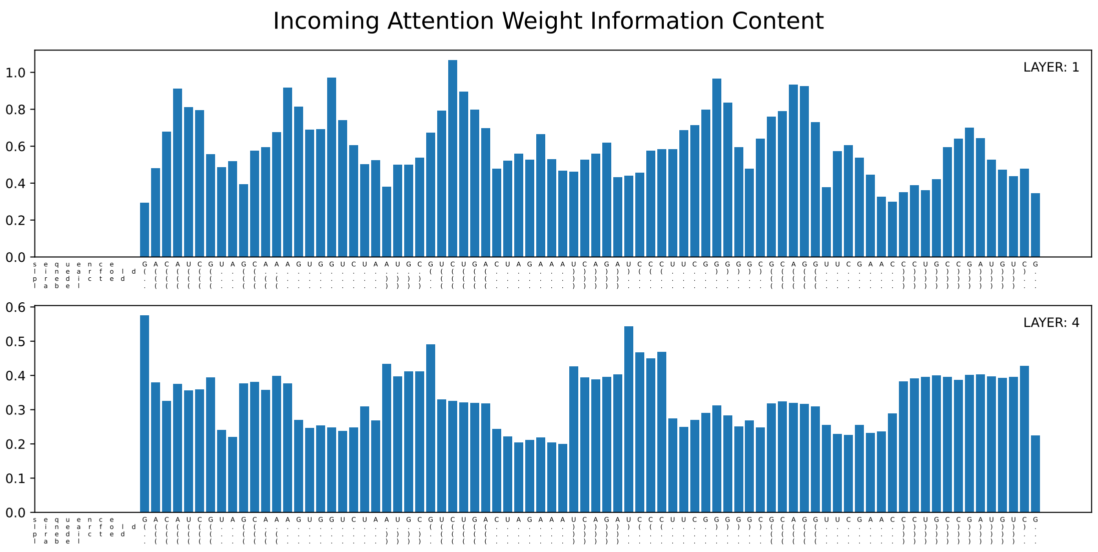


Figure 3.11: Incoming information content of an example fixed sequence (bpRNA_RFAM_1939), for the Attention Weights to each index. We see the incoming information content at layer 4, is higher in paired and fixed regions.

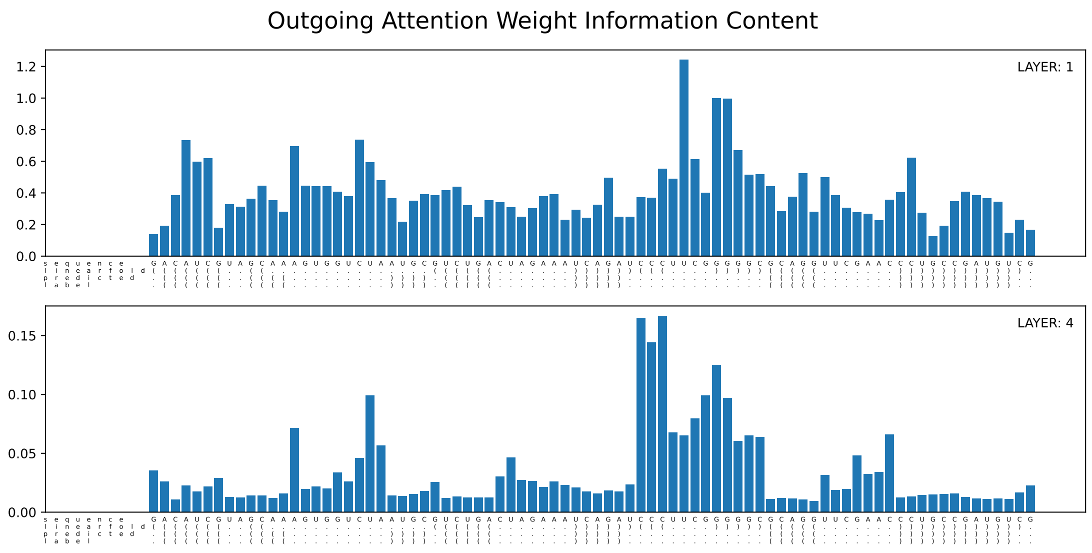


Figure 3.12: Outgoing information content of the same sequence as Figure 3.11, for the Attention Weights leaving from each index. This example features prominent larger outgoing information content at a fix (removed hairpin).

And with the observed entropy:

$$\hat{H}_i = -\sum_{j=0}^{n} w_{ij} \cdot \log_2(w_{ij})$$

We have:

$$IC_i = \log_2 n + \sum_{j=0}^{n} w_{ij} \cdot \log_2(w_{ij})$$

This is the information content for all of the attention weights going to each index, next we also calculate the information content of all of the attention weights coming from each index.

3.4.4.1 Outgoing Attention Weight information content

The Transformer Network softmax normalizes the attention weights going to each index, and so in order to compute the outgoing information content, we can normalize the outgoing weights.

$$Z_j = \sum_{i=0}^{n} w_{ij}$$

The normalizing constant Z_j is the sum of the attention weights to the position j, which can be found plotted in Figure 3.6. Our observed outgoing entropy becomes:

$$\hat{H}_j = -\sum_{i=0}^{n} \frac{w_{ij}}{Z_j} \cdot \log_2(\frac{w_{ij}}{Z_j})$$

And thus

$$IC_j = \log_2 n + \sum_{i=0}^{n} \frac{w_{ij}}{Z_j} \cdot \log_2(\frac{w_{ij}}{Z_j})$$

3.4.5 bpRNA Structure Labels

bpRNA offers programs in addition to their dataset that analyzes RNA sequences and dotbracket data. This includes programs that annotate regions of structure types including hairpin, loops, and multi-loops. To further extend our own information content-based analysis, we incorporate the use of these structure types, labeling both the Linearfold input and bpRNA-Fix output predictions. Computing the incoming information content at each position across the test data

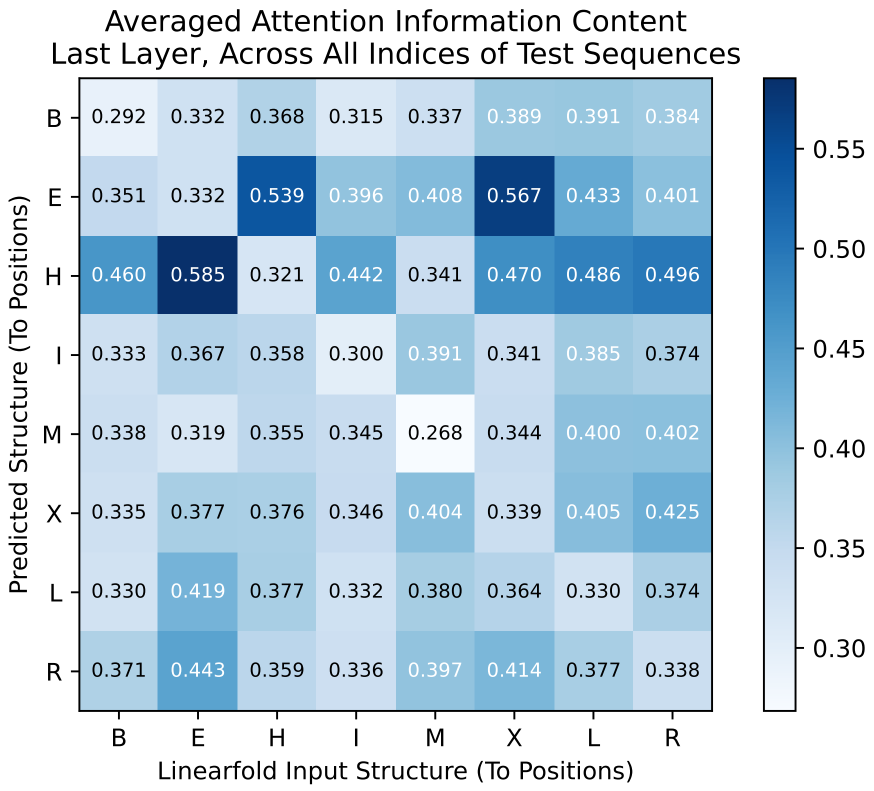


Figure 3.13: Average incoming information content in our model's last layer for each type of bpRNA structure label fix, going from Linearfold's input to bpRNA-Fix's predicted output. Fixes consisting of an end (E) to a hairpin (H) have high average information Content.

and binning them according to Linearfold input and bpRNA-Fix output structures, gives us the heatmap at Figure 3.13. Here we see what types of fixes causes the attention weights of our model to be the most non-uniform.

Filtering for examples featuring E→H fixes (from an end to a hairpin loop) we visualize the structure of a Linearfold input in 5B and the corresponding model output in 5C. The attention weights to the fixed position in the last layer 4 of our model were found to be enriched around the model output structures. We do the same for a longer example showcasing an example of an M→I fix (multiloop to internal loop), which also show the attention weights to the fix position are enriched around the output structure regions in Figure 3.15.

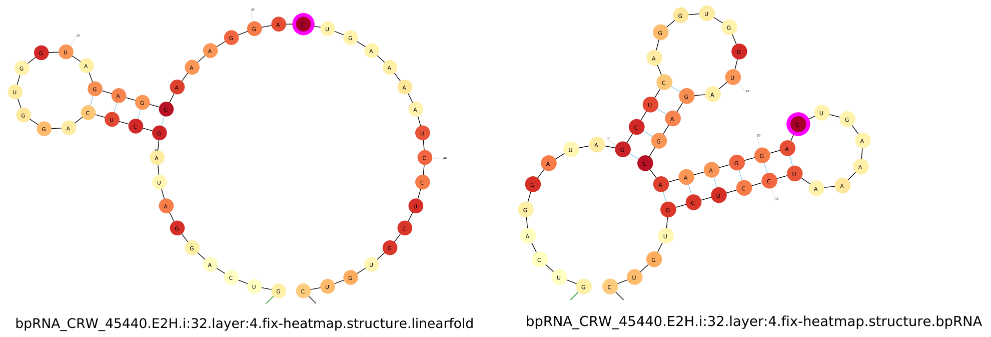

bpRNA_CRW_45440.E2H.i:32.layer:4.fix-heatmap.structure.linearfold
bpRNA_CRW_45440.E2H.i:32.layer:4.fix-heatmap.structure.bpRNA

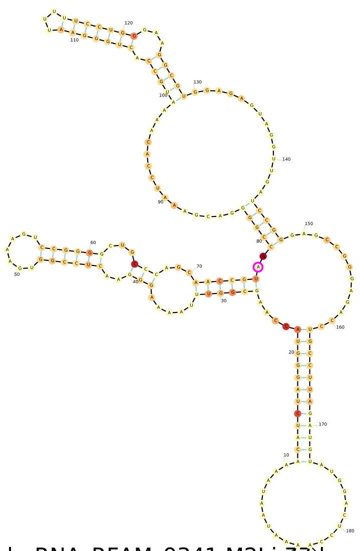
Figure 3.14: Left: Example test sequence as paired by Linearfold with the fix position (E to H) highlighted in magenta, and with the attention weights to this fix position heatmapped in red. Right: The resulting predicted output sequence from our model, with the same fix position highlighted and the same attention weights to this position, showing that attention across the Linearfold input dotbracket correspond to structures in the predicted output. Attentions between the hairpins can also be seen, suggesting the modeling of pseudoknot structures.

3.5 Discussion

We have shown that the introduction of a well-formed base pairing annotation output from another structure prediction program helps improve the performance of our own Transformer self-attention based network, adding parity information which our model ”scaffolds“ off of. This results in our bpRNA-Fix model making predictions that are better balanced than our baseline Attentionfold model without this additional input, and better overall performance than Linearfold and RNAfold, which provided our annotated inputs.

The RNA fixing task allows us to focus our analysis and visualization efforts on fixed positions, where our model output differs from the input dotbracket. Analysis of our model's attention weights using information content shows differentiation of roles between subsequent layers of the network, going from being more uniform to organizing around paired regions. We further show that stronger attentions at fixed positions are paid towards paired output regions. In this way, the model may not be ”thinking“ about the final output in a human sense, but still gives final output consideration in how it distributes its attention weights. There is even some evidence of pseudoknot structure modeling, which is beyond the scope of our secondary structure task, but likely does so in order to better perform its own task.

More generally, the idea that a previous program's output can improve the performance of


bpRNA_RFAM_9341.M2I.i:77.layer:4.fix-heatmap.structure.linearfold

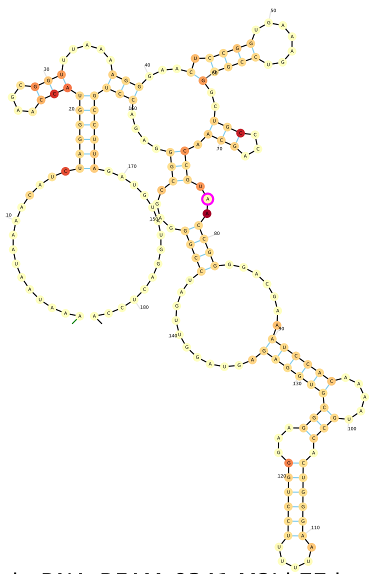

bpRNA_RFAM_9341.M2I.i:77.layer:4.fix-heatmap.structure.bpRNA

Figure 3.15: M→I fixes in Figure 3.13 are an enriched category for information content, here we show a more complex example also indicating that attention weights are distributed across stem structures in our predicted output.

deep learning based models can be readily applicable to other problem domains, especially those that utilize Transformer models on paired structure prediction tasks.

Chapter 4: Deep Learning for Dynamic Programming in RNA Structure Prediction

4.1 Introduction: Deep Nussinov

The secondary structure prediction of Ribonucleic acid (RNA) has been a long standing task in Bioinformatics, with several off-the-shelf programs such as CONTRAfold[12] or Linearfold[31] being available, which utilize thermodynamics-based model weights or energy parameters along with dynamic programming. Recently, models utilizing deep learning methods such as the self-attention Transformer model[70] have achieved great accuracy performances across Bioinformatics domains, from protein sequence structures[32] to DNA sequence classification[22] as examples.

Deep learning has also successfully been applied to RNA structure prediction[19, 67], getting current state-of-the-art results. However, one quirk often associated with deep learning based approaches is that their models may not necessarily output correctly balanced and matching base pairs or otherwise might not produce biologically plausible outputs (such as having base pair brackets being too close together). This is usually addressed with some form of post-processing after the model makes it's primary prediction[1]. This issue is not featured in previous dynamic programming approaches to RNA structure prediction, whose decision spaces avoid invalid outputs entirely. We combine the modeling benefits of the current deep learning models such as the Transformer, with an updated version of a classic dynamic programming algorithm for base pair matching in the Nussinov algorithm[46].

[1]For example by correcting the errors in the final output by running a simple parsing program that pushes to stack whenever a left bracket is read, pops the stack whenever a right bracket is read, and then disregards any right bracket that pops from an empty stack and any left bracket remaining on the stack at the end.

4.2 Methods

4.2.1 LSTM+Transformer Sequence Encoding

Utilizing the LSTM+Transformer baseline ”Attentionfold“ architecture from bpRNA-Fix, an RNA sequence input starting as a one-hot encoding is converted into a rich embedded representation using two bi-LSTM layers followed by four Transformer encoder layers. If the sequence length is n and our model embedding dimension size is d, then the output representation becomes a tensor of size $n \times d$. To measure how well two nucleotides in the sequence would pair together, we further introduce a simple feed-forward network that takes a $2d$ input and outputs two values representing paired and unpaired energy scores.[2] Thus, during decoding, whenever we want to query the pairing potential of two positions i and j, we would concatenate their d-sized embeddings together and run it through this feed-forward network.

For expediency at the cost of space, we can compute all pairings in parallel by stacking the $n \times d$ embedding row-wise and column-wise before concatenating them into an $n \times n \times 2d$ tensor representing all pairs of concatenations, before feeding this through the final feed-forward to get all $n \times n \times 2$ possible pairing scores. In practice, our model processes mini-batches of 16 sequences of up to length 200 nucleotides still fitting without issue on a 10GB GPU core, which helps speeds up our training and decoding.

4.2.2 The Nussinov Algorithm

The Nussinov dynamic programming algorithm[46] was presented in the late 1960’s originally as a means to maximize base pair matches. Instead of counting base pair matches, we can instead use the Nussinov algorithm to provide the set of base pair matches that maximize an energy score, provided an energy function that returns a score when queried for any two nucleotides. This is what we provide in our LSTM+Transformer pair scoring model, with energies being interpreted from neural network score values. This modified version of Nussinov’s algorithm is

[2]This could be simplified to a single score, if we were to let the “unpaired” score be zero. Currently, the unpaired score can have low or even negative values from our model.

thus shown here as Algorithm 1 below.[3][4]

Algorithm 1 DEEP NUSSINOV

```
Require: Model $m$: matrix of size $n \times n \times 2$ with paired/unpaired energy energy scores
Ensure: Score of highest scoring bracket structure according to model $m$
 1: for $i, j \leftarrow 0$ to $n$ do
 2:     $S[i,j] \leftarrow 0$
 3: end for
 4: for $span \leftarrow 1$ to $n$ do
 5:     for $i \leftarrow 0$ to $(n - span)$ do
 6:         $j \leftarrow i + span$
 7:         $S[i,j] \leftarrow \max\left(S[i,j-1] + m[i,j,0], \max_{k \in [i..(j-2)]}(S[i,k-1] + S[k+1,j-1] + m[k,j,1])\right)$
 8:     end for
 9: end for
10: return $S[n-1, n-1]$
```

4.2.3 Structured Perceptron Loss

During training, the model structure output is compared with the label and a site-wise Structured-Perceptron loss is accumulated across a mini-batch of sequences. That is, for the different possible outputs y_i at each position i in the sequence, and with s being a scoring function (the value output by the network for each output), we encourage the model to score the correct label output y_i^* higher than any other output y_i.

$$\forall y_i, s(y_i^*) - s(y_i) \geq 0$$

[3]To note, this only includes the forward decoding portion of the dynamic programming algorithm, in order to retrieve the output base pairs, we would need to maintain back pointers for every decision made by the algorithm and then backtrace.

[4]The algorithm's indices shown in Algorithm 1 are in base-0 indices, with inclusive-exclusive index start and stop positions respectively. This corresponds directly to the index semantics of many programming languages such as in Python.

During training and decoding the predicted output is selected by:

$$\hat{y_i} = \arg\max_{y_i} s_{\Delta}(y_i)$$

4.2.3.1 Structured SVM Loss and Loss-Augmented Decoding

To note, if we additionally enforced that the score of the correct label be higher than all other outputs by a margin $\Delta(y_i, y_i^*)$, then we would have a Structured SVM loss, and our training would thus be:

$$\forall y_i, s(y_i^*) - s(y_i) \geq \Delta(y_i, y_i^*)$$

During training time, we would predict outputs using the extra Δ margin:

$$\hat{y_i} = \arg\max_{y_i} s_{\Delta}(y_i) + \Delta(y_i, y_i^*)$$

The margin function Δ is designed to quantify the severity of incorrect predictions, with the end result being that the network would be encouraged to score correct predictions higher than incorrect predictions by at least this margin. The margin Δ is currently 0 for this work, but could be incorporated in the future. A simple loss-augmented decoding scheme could be utilized, increasing the margin of all incorrect predictions by one.[5]

4.2.4 Dataset and Training Time

The dataset from bpRNA-Fix provides an example train-validation-test dataset split for RNA secondary structure prediction. Because of our cubic time complexity in decoding runtime, we filter this dataset down to sequences of lengths up to 50, resulting in a split of (2897, 103, 90) train, validation, and test sequences respectively. The short sequences reduced our training time, allowing us to train 10 epochs with 20 evaluations to the validation per epoch (processing 28,970 training, 20,600 validation examples) in less than 21 hours. For initialization during training, we load a set of pretrained LSTM+Transformer model weights from a trained Attentionfold model,

[5]At the potential cost of computational expedience, much more complicated structured margin functions could be introduced instead for the goal of even better prediction accuracies. One idea would be to calculate how many label brackets would be violated or disturbed from an incorrect dotbracket prediction at each position. For example, mis-predicting a paired position as unpaired could shift an entire stem in the labeled dotbracket structure, or mispredicting a left-paired position as right-paired would affect at least two other different positions elsewhere in the sequence.

Model	Precision	Recall	F1	Seq Avg F1
RNAfold	54.26	**66.31**	59.69	53.85
Linearfold	69.31	59.37	63.95	51.09
Attentionfold (+Pos Enc)	36.63	20.69	26.45	19.79
Attentionfold (+LSTM)	40.84	45.47	43.03	43.47
bpRNA-Fix (+Linearfold)	52.16	49.24	50.66	47.81
bpRNA-Fix (+RNAfold)	65.13	62.08	63.57	58.44
Deep Nussinov	**69.52**	63.75	**66.51**	**58.97**

Table 4.1: Evaluation results for our models, on the bpRNA-Fix test set, filtered down for sequences of length ≤ 50. Precision, recall, and F1 measure the overall score across all base pair predictions in the test dataset. The sequence average F1 Score is the average F1 across each sequence. The Attentionfold and bpRNA-Fix models were originally trained on the full length 200 bpRNA-Fix dataset. Deep Nussinov was trained on the same dataset filtered down to sequences up to length 50, not making this an exact comparison.

which was originally trained on full bpRNA-Fix dataset.

4.3 Results

Table 4.1 shows our results on the test data, showing an overall improvement over off-the-shelf structure prediction programs and the bpRNA-Fix models. While Deep Nussinov is trained on a filtered version of the training set (filtering out sequences of length greater than 50), the bpRNA-Fix models were originally trained on the full unfiltered length-200 version of the training set, not making this an exact comparison. Figure 4.1 additionally shows corresponding error bars. Our Deep Nussinov model features the same encoder as the Attentionfold(+LSTM) model, but using dynamic programming producing structured outputs greatly helps our accuracy.[6]

4.4 Discussion

4.4.1 Cubic Time Complexity and Possible Linearization

The Nussinov algorithm runs in $O(n^3)$ time, utilizing $O(n^2)$ space in a scoring matrix. Because of this cubic time complexity, we have to limit ourselves to sequences up to length 50 in our

[6]Further performance improvements could be possible by incorporating RNAfold or Linearfold inputs such as in bpRNA-Fix.

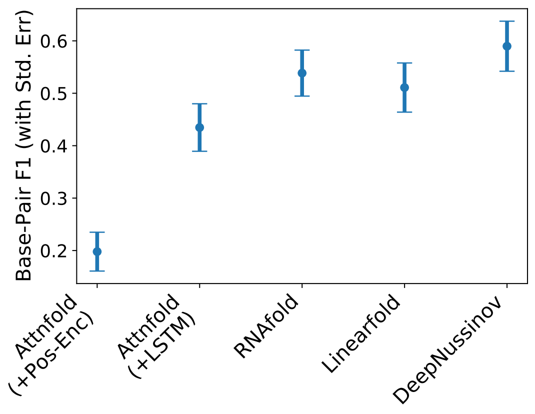


Figure 4.1: Standard Error plot of Deep Nussinov's performance on sequences up to length 50 in the bpRNA-Fix test set.

dataset. However, there exists literature with regards to inexact search within dynamic programming, mainly by changing the order of iteration of this algorithm to be bottom-up left-to-right, which then allows the use of beam search. This would end up being similar to a deep learning version Linearfold[31]. With b being the beam size constant, the linearized version would be an algorithm that runs in $O(n \cdot b^2)$, which could be further sped up to $O(n \cdot b \log(b))$, following Hong and Huang's[26] cube-pruning method.

4.4.2 Conclusions

We have shown that a deep learning architecture used for encoding a sequence into embeddings, followed by a dynamic programming decoding process allows us to constrain a model to only possibly produce valid dotbracket structures. This obviates the need to post-process the output or otherwise constrain the model to ensure balanced base pairings, and allows the model to focus it's training and learning efforts into improving its energy pairing model, without worrying about output base pair parities. Furthermore, the dynamic programming algorithm can be tuned to prevent pairings of certain close distances, as close base pair distances of one or two are very rare in real life biological contexts.

We show strong initial results by training our model on a filtered down version of bpRNA-Fix's dataset, and we compare to bpRNA-Fix models showing an improvement. However, as those models were trained on the full training set, this makes our comparison not quite exact.

Further evaluation work is needed, training comparative bpRNA-Fix models also on the length 50 dataset.

This work additionally features avenues for continued improvements, whether it is linearizing the algorithmic runtime to practically handle scaling up to longer sequences, larger neural network sizes, or loss-augmented decoding for improved training. There is also a question of visualizing the deep Nussinov model similar to work done in bpRNA-Fix, of what the model learns about RNA structure and base pairing. Dynamic programming decoding approaches such as deep Nussinov represent a principled approach to incorporating structure to deep learning.

Chapter 5: Deep Learning for Improving Protein Tertiary Structure Prediction

5.1 Introduction

Protein structure prediction is a longstanding and challenging problem in computational biology, involving the prediction of three-dimensional macromolecule structures from folding linear chains of amino acid sequences[6]. Like RNA, the structure of the sequence determines its overall function, and determining this structure can be challenging as heavily mutated sequences can still fold into the same structures. This task among many others in bioinformatics has gained renewed interest with increasing computational power and the usage of deep learning techniques.

In this and other biological sequential domains, a common utilized method is the leveraging of similar protein sequences whenever available, and aligning them together as a Multiple Sequence Alignment (MSA), which is used as the input to the program. This helps in part because homologous sequences will have similar characteristics and behavior, and mutations will occur largely in either unimportant regions or otherwise in structure preserving ways. An MSA would thus help a program identify important regions of the reference input sequence as well as co-occurring mutations. To maximize this information, protein structure prediction approaches incorporate the use of large MSAs[53, 32] (e.g. of sizes greater than 1,000) formed as a result of a database search and alignment, utilizing programs such as hhblits[54]. The quality and size of this MSA input has great effect on their downstream performance[20, 33], but it is precisely protein families with numerous database entries (for example in pdb[63]) that already have much research attention; a given protein sequence that has many homologous database entries is thus more likely to have known 3D structures. A novel or under-researched protein sequence with few existing database entries would not work as well with the current protein structure prediction paradigm. In this context, improving the alignment quality of a given shallow MSA would be important to help maximize performance in a predictive model.

We predict low resource protein structures by utilizing a joint folding and alignment algorithmic approach based on Turbofold II[65] and LinearTurboFold[39], which were originally inspired from Sankoff's algorithm[55]. This iterative approach computes match scores from the structures produced by the model to realign the input MSA as well as to compute an external information; this external information and realigned MSA is fed back into the model to produce refined output structures.

In order to produce multiple structures for alignment, an initial approach would be to re-run the model in its entirety to predict each input MSA sequence. However as this is computationally expensive, we additionally present a model that can output multiple structure predictions in parallel corresponding to the input sequences in the MSA. This makes computationally feasible our entire approach of jointly folding and aligning a shallow MSA (≤ 10 sequences) refined over several iterations.

5.2 Methods

5.2.1 Distance Map Prediction

Protein Folding is commonly formulated as a contact map prediction task[10] as a coarse intermediary prediction to be refined by a downstream program into a final 3D structure. For a protein sequence of length L, a predicted contact map is an $L \times L$ matrix holding binary contact or non-contact predictions. Each cell at position (i, j) represents whether the amino acids at i and j are in contact. Recently, this has been task has been generalized to the prediction of binned distance values[73, 74], in which each cell of the contact map now predicts between a set of distance values; this granularity has been found to improve downstream 3D modeling. Yang Jianyi et al. defines these starting from a minimum bin of 2.0 Angstrom, and each subsequent bin representing 0.5 Angstrom increments until a final bin of "no contact" representing distances beyond 20 Angstroms, resulting in 37 possible predicted values being stored in the distance map. They also go further and additionally use their model to predict inter-residue angles as well, which also take the form of an $n \times n$ matrix for every pair of positions in the sequence. These predicted distance maps and angle maps are then later used as constraints in physics optimization packages such as trRosetta[13], to produce 3D structures. To constrain the scope of this paper we will focus on improving the predictions of distance maps instead of handling the full 3D structures, with the understanding that improved distance map accuracies correlates with improved final structures. We will also use the phrase distance map and contact map interchangeably, as any given distance map can always be binarized back into a contact map.

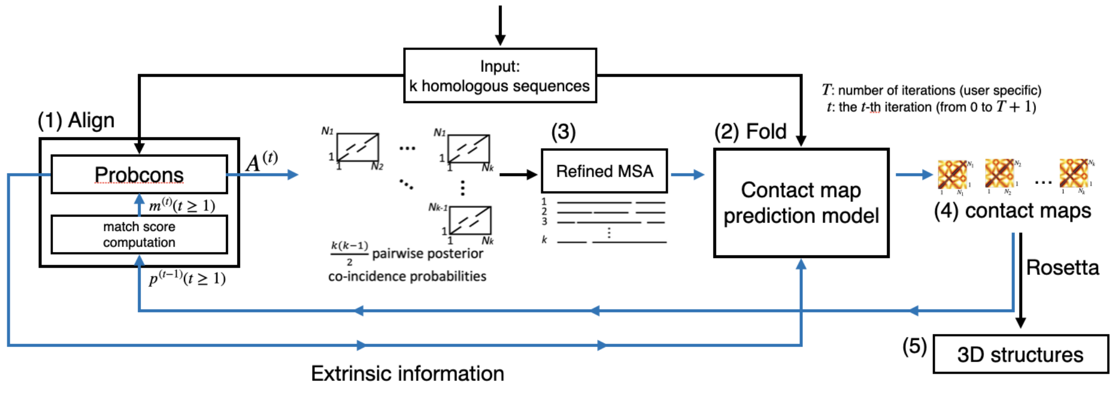


Figure 5.1: High level overview of the TurboProFold process. An input of k homologous or similar sequences are provided to our Align and Folding modules. 1) The alignment module consists of a match score computation and Probcons, in the initial iteration without contact maps predictions, the input sequences are either aligned with Probcons or in our case given as aligned. In subsequent iterations, a set of contact maps would be made available and these are analyzed using a match score computation before being input to Probcons. 2) The folding module consists of a contact map prediction model that takes an MSA as input and outputs a set of k contact maps. We utilize a publicly available deep learning model based on Yang et al. 2020[74] for this purpose. 3) Our alignment module uses the match scores to re-align the input sequences and compute an extrinsic information, both of which get sent to the folding module to help improve its contact map output predictions. 4) The folding module produces a set of contact maps, which can be sent back to the alignment module for another iteration of TurboProFold, or the contact map of the reference sequence can be submitted to Rosetta (5) to finalize 3D structures.

5.2.2 Protein Distance Map Prediction Model

We utilize the TAPE Proteins Library[52], which provides a PyTorch implementation of Yang Jianyi's protein folding model[74] as well as their training and test dataset. This model receives an MSA as input but was originally designed to assume that the first sequence of the MSA was the input query sequence, called the reference sequence. The model extracts one-dimensional sequence features from the reference sequence, which are then concatenated row and column-wise to the extracted two-dimensional MSA features taken from the inverse of the covariance matrix computed from the co-occurrence statistics of the MSA. These features are fed through 60 layers of ResNet[64] residual CNN layers, before a final set of convolutions for distance and angle predictions.

We begin with this baseline model and modify what kind of inputs it reads as well as what

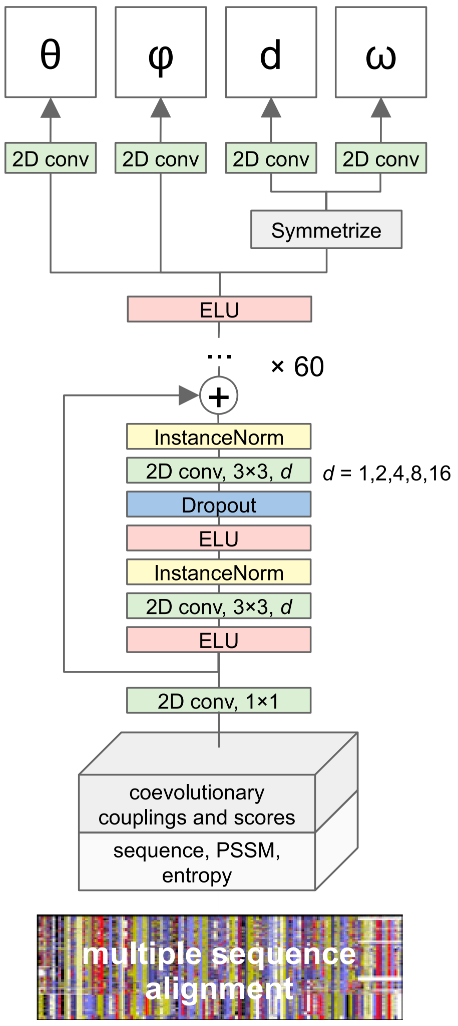


Figure 5.2: Figure of Yang Jianyi's baseline model[74] taken from their paper. Starting with an input MSA, one-dimensnional and two-dimensional features are extracted and concatenated together into a 2D matrix. This is then fed into several residual-CNN layers before predicting distance and inter-angle maps, to be fed as constraints in trRosetta[13] for final 3D structure predictions.

kind of outputs it produces. We first allow the reading of gaps in the reference sequence of the MSA, which may get added as the MSA becomes realigned. We also output a contact map prediction for every sequence in the MSA, allowing us to expediently produce structure alignments

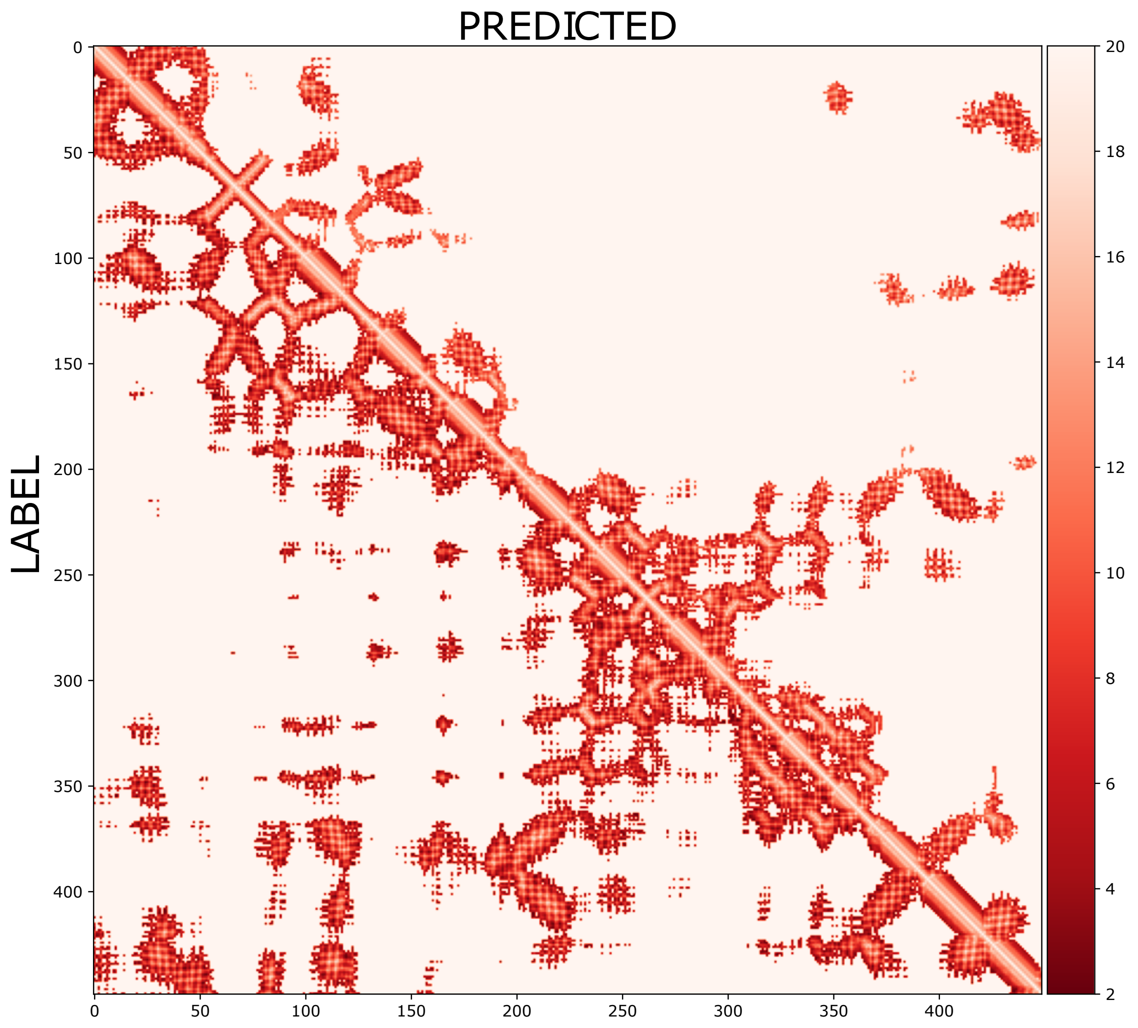


Figure 5.3: Distance map visualization produced by our baseline model, on a validation sequence example. The predicted distance bins are converted back into Angstroms and compared with the labeled structure, and distances ≥ 20 Angstroms are binned together. Contacts closer to the diagonal of this plot are more local in scope, and contacts towards the bottom left and upper right corners are more global in scope. Alpha helices (contacts close and parallel to the diagonal) and beta sheets (contacts close and perpendicular to the diagonal) can be seen visualized. The Yang Jianyi model is primarily built upon stacked Resnet CNN layers, which does well at modeling local structures, but struggles with identifying global contacts, which remains sparse compared to the label.

among the outputs, by taking the second-to-last rich embedded output of the model [1] and keep the corresponding rows and columns for each sequence's non-gap character positions. This can be done just with bookkeeping and in-place indexing using PyTorch, without requiring any additional space. For each sequence of the MSA, this produces corresponding two-dimensional embeddings and subsequent distance and angle map predictions.[2]

5.2.3 Match Score and Extrinsic Information

The output distance maps are aligned, first by using a series of computed sequence-pair match scores, which measures structural similarity for position pairs between a pair of sequences. The sequence-pair match scores are then input into Probcons[11], producing pairwise alignments of sequences within the MSA as well as an overall realignment of the MSA itself. The pairwise alignments are then used to compute an extrinsic information, modeling the proclivity for base pairing induced from the other sequences in the MSA. Both the match score and extrinsic information are detailed in LinearTurboFold[39].

This extrinsic information is incorporated into the model by concatenating it with the second-to-last layer. As extrinsic information is a set of scalar values, the concatenation step increases the dimensionality of the model's embedding by one, before a feed-forward network is applied turning this embedding back to model dimension size. The TurboProFold joint folding and aligning process thus utilizes both this extrinsic information and the realigned input MSA as refined inputs in the next iteration.

5.2.4 Dataset and Training

We utilize the same TAPE training and validation dataset that the baseline model was trained on. Starting with a trained instance of the original model, we process the MSAs in the training set, producing the first up to 100 distance maps and corresponding first-round extrinsic information. We then apply one epoch of training with this training set and extrinsic information, producing

[1]From Figure 5.2 this is the output right after the final ELU, before the last 2D Convolution layers output the different distance and angle map predictions.

[2]Informally comparing this method to running the entire model for each sequence of the MSA, done by swapping each sequence of the MSA with the first reference sequence before rerunning, we find a substantial speed up. In order to produce 100 contact maps for each of the 15,000 MSAs from the TAPE dataset, what would take a projected two months becomes about 4 hours to finish using a compute cluster.

	Very Short $[1,6)$		Short $[6,12)$		Medium $[12,24)$		Long $[24,)$		All $[1,)$	
	Top-L	Overall	Top-L	Overall	Top-L	Overall	Top-L	Overall	Top-L	Overall
TurboProFold										
Iteration 0	96.26	56.66	07.71	05.73	06.40	04.70	05.42	03.33	96.21	12.63
Iteration 1	96.32	56.62	**07.98**	**05.80**	**06.87**	**04.92**	**06.10**	**03.50**	96.30	12.51
Iteration 2	96.46	56.70	07.84	05.78	06.52	04.76	05.48	03.39	96.42	12.55
Iteration 3	96.46	56.66	07.85	05.79	06.52	04.77	05.46	03.39	96.43	12.58
...										
Iteration 9	**96.56**	**56.79**	07.71	05.78	06.54	04.82	05.49	03.42	**96.53**	**12.72**
AlphaFold2*	-	56.87	-	06.52	-	06.56	-	04.24	-	13.89

Table 5.1: Evaluation experiment done with the CAMEO-L dataset, with input MSA size $\leq$ 10. The precision of the Top-L most confident predictions are listed, along with the overall precision for each distance category. The MSA realignment and extrinsic information generated from the TurboProFold model improves contacts at each distance, most benefiting very short contacts. Short, medium, and long range contacts converge by one round of the joint folding and aligning process, while very short contacts continue to improve from iteration to iteration. We also show results from running AlphaFold2, modified to prevent it from searching for additional homologous sequences to augment its input. AlphaFold2 outputs a 3D structure, that we convert to a contact map for our evaluation. Since we only have the contact predictions, we do not have a notion of its top-L most confident predictions, and so we only show its overall precision.

our fine-tuned model. Our subsequent test data for our experiments comes from a prepared CAMEO dataset with downsampled MSAs, called CAMEO-L prepared by Wang et al 2022[72]. Table 5.1 shows our subsequent results.

5.3 Results

We use our TurboProFold method to predict the structures of the CAMEO-L dataset. We use our joint fold-and-align pipeline to produce several iterations of these structures, and then do a comparison across iterations. In order to evaluate the reference sequence contact map for each test example, we divide up different distance categories (short, medium, long, etc) representing the distance along the input sequence. For example, for the contact between two positions i and j in the sequence, $|i-j|$ would measure the distance between them. If this distance is 10, then this would be a “short” contact, or if this distance is 30, then this would be a “long” contact. Given L the length of a sequence, the Top-L precision represents the precision of the L most confident contact predictions made by the model. We showcase this evaluation along with the overall precision across every contact within a distance category.

When our model is run for the first time (using the given input MSA and an extrinsic information set to zero values), this is called iteration 0. We show in Table 5.1 that iteration 1 yields the most benefit for short, medium, and long contacts, and that subsequent iterations improve very short contacts (i.e. contact distances less than 6). This suggests that our CNN-based model ultimately improves the most on local secondary structures from iteration to iteration (alpha helices and beta sheets).

For reference, our results are compared with AlphaFold2[32]. After inputting the shallow MSA, we additionally disable its calls to hhblits[54], disallowing it from searching its accompanying protein sequence databases for additional homologous sequences to augment to the MSA. We limit AF2 in this way to try and simulate how it would perform in a shallow MSA context. Since AlphaFold2 directly outputs a 3D structure, we then convert this back into a contact map in order to compare with our outputs.[3] Shown at the bottom of Table 5.1, AlphaFold2 yields a very strong baseline that we do not surpass.

5.3.1 Alignment Evaluation

We additionally evaluate the improvement in our alignments after running one round of TurboProFold on the OXBench[51] dataset, which can be seen in Table 5.2. Taking a pair of sequences to align, we first run our folding model to produce contact maps, for which we produce match score information for. We then use ProbCons[11] from our alignment module as our aligner for these sequences, and we compare using it as a baseline aligner versus additionally providing our computed match scores. We find TurboProFold generally does improve our alignments, but we show the most improvement when the sequences being aligned have low identity. For sequences that look dissimilar with high sequence distance (having many structure preserving mutations), the inclusion of match scores most improves ProbCons's ability to align.

5.4 Discussion

We present a joint folding and aligning algorithm for protein distance map tertiary structure prediction, where multiple predictions from an existing prediction model can be used to realign the MSA and calculate an extrinsic information useful for iteratively improving performance. In

[3] AlphaFold2 has an additional advantage when converting its 3D structures into contact maps, since these contact maps will naturally not violate any real-world constraints on distances.

Sequence Identity	ProbCons ppv	ProbCons sen	+match score ppv	+match score sen	improvement ppv	improvement sen
>0.50	92.4	92.2	92.6	92.4	0.2	0.2
0.40 - 0.50	83.9	83.6	84.9	84.6	1.1	1.0
0.30 - 0.40	78.4	78.2	79.8	79.5	1.4	1.3
<0.30	43.0	43.1	46.8	46.4	**3.8**	**3.3**

Table 5.2: Our alignment result improvements after using one round of TurboProFold, on the OXBench[51] dataset. Using ProbCons[11] by itself to perform alignments versus additionally adding our computed match scores shows improvements across the different sequence identity categories. When the sequence identity is very low (less than 30%) is when our match score most improves ProbCons's ability to align.

practice, predicting each contact map from an MSA can be computationally expensive, so we also show a decoding method to cheaply produce several or even all contact maps from an MSA using only the second-to-last embedded layer of the model. To showcase our idea we adapt an already existing protein folding model, demonstrating that the folding and alignment strategy can be used to improve performance especially in a shallow MSA context, maximizing the available information from homologous sequences. We show that our model benefits from this approach, especially in very short local contacts, as well as medium and longer distance contacts as well. Focusing on evaluating how well our alignments improve after one iteration of TurboProFold, we find that by first folding then aligning, our alignments generally improve, and we show the most improvement for aligning sequences with low sequence identity. Finally, the joint folding and aligning approach is a general idea and can be applied to use other models. For example, AlphaFold2 could be used instead of our Resnet-based model, allowing the TurboProFold method to improve AlphaFold2's structure prediction results.

Chapter 6: Summary

We have presented deep learning approaches along with related methods for parsing languages and folding RNA and Protein structures. A variety of deep learning architectures were ultimately used from LSTMs modeling sequential information, to Transformers to directly encode inter-sequence relationships[1], to CNNs to model structures local in scope. But beyond this, we further show how deep learning-related limitations interact with domain-specific challenges, and the different methods we use to address these.

We have shown that direct unstructured decoding methods can be simple and computationally expedient, but can suffer from incorrect or implausible predictions. This can be addressed by some post processing, by enriching the input with more structured information (bpRNA-Fix), or by using the prediction as an intermediary in a pipeline for a downstream final prediction (protein contact map prediction). Furthermore, we have also shown that dynamic programming algorithms can provide us the ability to directly output structured predictions ultimately giving us better accuracies (span-parsing[2] with CKY or shift-reduce, and deep nussinov). Instead of an ad-hoc post processing, this represents a principled way to constrain a model to only generate valid outputs. Furthermore, if certain trade-offs with search quality are willing to be made (slow and exact vs fast and inexact), then our span parsing work with cube-pruned beam search demonstrates how we can make dynamic programming methods achieve fast runtime complexity.

While seemingly underutilized even when available, dynamic programming approaches for a given structure prediction problem may not exist or be formulated as of yet, leaving space for new research. In the case of TurboProFold's contact map predictions for protein structure, the output contact maps are still unconstrained, making it possible for the predicted distances to suggest implausible 3D structures. Work for structured outputs for this problem might be developed in the future, but in general researchers should try to consider a structured decoding if possible. TurboProFold also represents a method to maximize the benefit of given information from a dataset. Here, a protein folding model benefitted because we took the structure alignment

[1]The Transformer self-attention mechanism relates all pairs of sequence tokens in one layer of encoding. This serves as a deep learning analog to a bilexical grammar[17].

[2]Span parsing is simple yet effective, such that more expressive and complex linguistic formalisms can be handled beyond just syntactic constituency parsing, such as Combinatory Categorical Grammars[16, 34].

on its outputs, producing both an extrinsic information utilized as additional input back into the model and a realigned input MSA. Although benefits can be measured after one iteration, this process can be iterated arbitrarily many times until the outputs converge.

We also presented methods to visualize and interpret models once they achieve competitive accuracy. For bpRNA-Fix, we showed that the parity (the "evenness") of the dotbrackets output by our model improved when an already paired input was added. The underlying Transformer self-attention network evidently struggled less to output a matching set of base pairs when a suggested pairing was provided. We could focus on fixes as the particular parts of a dotbracket output by our model that differed from this input, and showed that in certain types of fix cases, the model attends strongly to the eventual output structure. These are shown in addition to other notable behaviors such as having differentiated behavior from layer to layer, being generally more uniform in the initial layers and peakier around structures in the latter layers.

Finally, bpRNA-Fix represents a general idea that could be applied to problem domains not just within biological or linguistic contexts, but whenever a cheap or easy-to-compute source of annotation can be available.[3] Utilizing a previous model as an input annotation source can be seen essentially as a form of fine-tuning, but instead of just leveraging large established deep learning models, enriched inputs can come from a wider variety of sources. Having indirect access to a simple or fast prediction model can improve performance and additionally can allow the training of smaller models upon less available data.

Both natural language processing and bioinformatics have had recent exciting developments and progress with the advancement of deep learning, and this thesis is in part a document of this era. Until future AI methods can output their own structured predictions and interpret their own results, we predict that algorithms and computer scientists will still yet be needed.

[3]For example, a deep learning-based chess-playing AI system could still benefit from taking the move recommendations from a simple chess program coded using a minimax algorithm run out to a shallow depth, incorporating its rough set of recommendations. Or for another example, a deep learning-based weather prediction model could still benefit from the output of traditional forecasting models which have already been produced by weather stations.

Bibliography

[1] Ngo Xuan Bach, Nguyen Le Minh, and Akira Shimazu. A reranking model for discourse segmentation using subtree features. In *Proceedings of the 13th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 160–168. Association for Computational Linguistics, 2012.

[2] Anders Björkelund, Agnieszka Faleńska, Wolfgang Seeker, and Jonas Kuhn. How to train dependency parsers with inexact search for joint sentence boundary detection and parsing of entire documents. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1924–1934, 2016.

[3] Marc Brysbaert, Michaël Stevens, Paweł Mandera, and Emmanuel Keuleers. How many words do we know? practical estimates of vocabulary size dependent on word definition, the degree of language input and the participant's age. *Frontiers in Psychology*, 7, 2016.

[4] David Chiang. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–208, 2007.

[5] Do Kook Choe and Eugene Charniak. Parsing as language modeling. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 2331–2336, 2016.

[6] Thomas E Creighton. Protein folding. *Biochemical journal*, 270(1):1, 1990.

[7] James Cross and Liang Huang. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Austin, Texas, November 2016. Association for Computational Linguistics.

[8] Padideh Danaee, Mason Rouches, Michelle Wiley, Dezhong Deng, Liang Huang, and David Hendrix. bprna: large-scale automated annotation and analysis of rna secondary structure. *Nucleic acids research*, 46(11):5381–5394, 2018.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[10] Pietro Di Lena, Ken Nagata, and Pierre Baldi. Deep architectures for protein contact map prediction. *Bioinformatics*, 28(19):2449–2457, 2012.

[11] Chuong B Do, Mahathi SP Mahabhashyam, Michael Brudno, and Serafim Batzoglou. Probcons: Probabilistic consistency-based multiple sequence alignment. *Genome research*, 15(2):330–340, 2005.

[12] Chuong B. Do, Daniel A. Woods, and Serafim Batzoglou. CONTRAfold: RNA secondary structure prediction without physics-based models. *Bioinformatics*, 22(14):e90–e98, 07 2006.

[13] Zongyang Du, Hong Su, Wenkai Wang, Lisha Ye, Hong Wei, Zhenling Peng, Ivan Anishchenko, David Baker, and Jianyi Yang. The trrosetta server for fast and accurate protein structure prediction. *Nature protocols*, 16(12):5634–5651, 2021.

[14] Greg Durrett and Dan Klein. Neural CRF parsing. *arXiv preprint arXiv:1507.03641*, 2015.

[15] Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. Recurrent neural network grammars. *arXiv preprint arXiv:1602.07776*, 2016.

[16] Jason Eisner. Efficient normal-form parsing for combinatory categorial grammar. *arXiv preprint cmp-lg/9605038*, 1996.

[17] Jason Eisner. Bilexical grammars and their cubic-time parsing algorithms. In *Advances in probabilistic and other parsing technologies*, pages 29–61. Springer, 2000.

[18] Daniel Fried, Mitchell Stern, and Dan Klein. Improving neural parsing by disentangling model combination and reranking effects. In *Proceedings of the Association for Computational Linguistics*, 2017.

[19] Laiyi Fu, Yingxin Cao, Jie Wu, Qinke Peng, Qing Nie, and Xiaohui Xie. UFold: fast and accurate RNA secondary structure prediction with deep learning. *Nucleic Acids Research*, 50(3):e14–e14, 11 2021.

[20] Hiroyuki Fukuda and Kentaro Tomii. Deepeca: an end-to-end learning framework for protein contact prediction from a multiple sequence alignment. *BMC bioinformatics*, 21(1):1–15, 2020.

[21] Michael Hahn. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171, 2020.

[22] Shujun He, Baizhen Gao, Rushant Sabnis, and Qing Sun. Nucleic transformer: Deep learning on nucleic acids with self-attention and convolutions. *bioRxiv*, 2021.

[23] Hugo Hernault, Helmut Prendinger, David A DuVerle, Mitsuru Ishizuka, and Tim Paek. Hilda: a discourse parser using support vector machine classification. *Dialogue and Discourse*, 1(3):1–33, 2010.

[24] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[25] Ivo L. Hofacker. Vienna RNA secondary structure server. *Nucleic Acids Research*, 31(13):3429–3431, 07 2003.

[26] Juneki Hong and Liang Huang. Linear-time constituency parsing with rnns and dynamic programming. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 477–483, 2018.

[27] Kexin Huang, Cao Xiao, Lucas M Glass, and Jimeng Sun. MolTrans: Molecular Interaction Transformer for drug–target interaction prediction. *Bioinformatics*, 37(6):830–836, 10 2020.

[28] Liang Huang and David Chiang. Forest rescoring: Fast decoding with integrated language models. In *Proceedings of ACL 2007*, Prague, Czech Rep., 2007.

[29] Liang Huang, Suphan Fayong, and Yang Guo. Structured perceptron with inexact search. In *Proceedings of NAACL*, 2012.

[30] Liang Huang and Kenji Sagae. Dynamic programming for linear-time incremental parsing. In *Proceedings of ACL 2010*, Uppsala, Sweden, 2010.

[31] Liang Huang, He Zhang, Dezhong Deng, Kai Zhao, Kaibo Liu, David A Hendrix, and David H Mathews. Linearfold: linear-time approximate rna folding by 5'-to-3' dynamic programming and beam search. *Bioinformatics*, 35(14):i295–i304, Jul 2019.

[32] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[33] Shaun M Kandathil, Joe G Greener, and David T Jones. Prediction of interresidue contacts with deepmetapsicov in casp13. *Proteins: Structure, Function, and Bioinformatics*, 87(12):1092–1099, 2019.

[34] Yoshihide Kato and Shigeki Matsubara. A new representation for span-based ccg parsing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10579–10584, 2021.

[35] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[36] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[37] Eliyahu Kiperwasser and Yoav Goldberg. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *CoRR*, abs/1603.04351, 2016.

[38] Nguyen Quoc Khanh Le, Quang-Thai Ho, Trinh-Trung-Duong Nguyen, and Yu-Yen Ou. A transformer architecture based on BERT and 2D convolutional neural network to identify DNA enhancers from sequence information. *Briefings in Bioinformatics*, 22(5), 02 2021. bbab005.

[39] Sizhen Li, He Zhang, Liang Zhang, Kaibo Liu, Boxiang Liu, David H Mathews, and Liang Huang. Linearturbofold: Fast folding and alignment for rna homologs with applications to coronavirus. *bioRxiv*, pages 2020–11, 2021.

[40] Weizhong Li and Adam Godzik. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22(13):1658–1659, 2006.

[41] Jiangming Liu and Yue Zhang. Shift-reduce constituent parsing with neural lookahead features. *arXiv preprint arXiv:1612.00567*, 2016.

[42] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[43] Haitao Mi and Liang Huang. Shift-reduce constituency parsing with dynamic programming and pos tag lattice. In *Proceedings of NAACL 2015*, 2015.

[44] Jean-Baptiste Michel, Yuan Kui Shen, Aviva Presser Aiden, Adrian Veres, Matthew K Gray, Google Books Team, Joseph P Pickett, Dale Hoiberg, Dan Clancy, Peter Norvig, et al. Quantitative analysis of culture using millions of digitized books. *science*, 331(6014):176–182, 2011.

[45] Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*, 2017.

[46] Ruth Nussinov and Ann B Jacobson. Fast algorithm for predicting the secondary structure of single-stranded rna. *Proceedings of the National Academy of Sciences*, 77(11):6309–6313, 1980.

[47] J. Padarian, A. B. McBratney, and B. Minasny. Game theory interpretation of digital soil mapping convolutional neural networks. *SOIL*, 6(2):389–397, 2020.

[48] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International Conference on Machine Learning*, pages 4055–4064. PMLR, 2018.

[49] Matthew E Peters, Mark Neumann, Luke Zettlemoyer, and Wen-tau Yih. Dissecting contextual word embeddings: Architecture and representation. *arXiv preprint arXiv:1808.08949*, 2018.

[50] Zhuwei Qin, Fuxun Yu, Chenchen Liu, and Xiang Chen. How convolutional neural network see the world-a survey of convolutional neural network visualization methods. *arXiv preprint arXiv:1804.11191*, 2018.

[51] GPS Raghava, Stephen MJ Searle, Patrick C Audley, Jonathan D Barber, and Geoffrey J Barton. Oxbench: a benchmark for evaluation of protein multiple sequence alignment accuracy. *BMC bioinformatics*, 4(1):1–23, 2003.

[52] Roshan Rao, Nicholas Bhattacharya, Neil Thomas, Yan Duan, Peter Chen, John Canny, Pieter Abbeel, and Yun Song. Evaluating protein transfer learning with tape. *Advances in neural information processing systems*, 32, 2019.

[53] Roshan M Rao, Jason Liu, Robert Verkuil, Joshua Meier, John Canny, Pieter Abbeel, Tom Sercu, and Alexander Rives. Msa transformer. In *International Conference on Machine Learning*, pages 8844–8856. PMLR, 2021.

[54] Michael Remmert, Andreas Biegert, Andreas Hauser, and Johannes Söding. Hhblits: lightning-fast iterative protein sequence searching by hmm-hmm alignment. *Nature methods*, 9(2):173–175, 2012.

[55] David Sankoff. Simultaneous solution of the rna folding, alignment and protosequence problems. *SIAM journal on applied mathematics*, 45(5):810–825, 1985.

[56] Kengo Sato, Manato Akiyama, and Yasubumi Sakakibara. Rna secondary structure prediction using deep learning with thermodynamic integration. *Nature communications*, 12(1):1–9, 2021.

[57] Tianze Shi, Liang Huang, and Lillian Lee. Fast(er) exact decoding and global training for transition-based dependency parsing via a minimal feature set. In *Proceedings of EMNLP 2017 (to appear)*, 2017.

[58] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. *CoRR*, abs/1704.02685, 2017.

[59] Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. Parsing with compositional vector grammars. In *Proceedings of the Association for Computational Linguistics*, volume 1, pages 455–465. Association for Computational Linguistics, 2013.

[60] Mitchell Stern, Jacob Andreas, and Dan Klein. A minimal span-based neural constituency parser. In *Proceedings of the Association for Computational Linguistics*, 2017.

[61] Mitchell Stern, Jacob Andreas, and Dan Klein. A minimal span-based neural constituency parser (code base). `https://github.com/mitchellstern/minimal-span-parser`, 2017.

[62] Mitchell Stern, Daniel Fried, and Dan Klein. Effective inference for generative neural parsing. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 1695–1700, 2017.

[63] Joel L Sussman, Dawei Lin, Jiansheng Jiang, Nancy O Manning, Jaime Prilusky, Otto Ritter, and Enrique E Abola. Protein data bank (pdb): database of three-dimensional structural information of biological macromolecules. *Acta Crystallographica Section D: Biological Crystallography*, 54(6):1078–1084, 1998.

[64] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[65] Zhen Tan, Yinghan Fu, Gaurav Sharma, and David H Mathews. Turbofold ii: Rna structural alignment and secondary structure prediction informed by multiple homologs. *Nucleic acids research*, 45(20):11570–11581, 2017.

[66] Masaru Tomita, editor. *Generalized LR Parsing*. Kluwer Academic Publishers, 1991.

[67] Raphael J. L. Townshend, Stephan Eismann, Andrew M. Watkins, Ramya Rangan, Maria Karelina, Rhiju Das, and Ron O. Dror. Geometric deep learning of rna structure. *Science*, 373(6558):1047–1051, 2021.

[68] Alan Mathison Turing. The chemical basis of morphogenesis. *Bulletin of mathematical biology*, 52(1):153–197, 1990.

[69] Alan Mathison Turing. The applications of probability to cryptography, c. 1941. *UK National Archives, HW*, 25:37, 2012.

[70] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[71] Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pages 2773–2781, 2015.

[72] Qin Wang, Jiayang Chen, Yuzhe Zhou, Yu Li, Liangzhen Zheng, Sheng Wang, Zhen Li, and Shuguang Cui. Contact-distil: Boosting low homologous protein contact map prediction by self-supervised distillation. 2022.

[73] Jinbo Xu. Distance-based protein folding powered by deep learning. *Proceedings of the National Academy of Sciences*, 116(34):16856–16865, 2019.

[74] Jianyi Yang, Ivan Anishchenko, Hahnbeom Park, Zhenling Peng, Sergey Ovchinnikov, and David Baker. Improved protein structure prediction using predicted interresidue orientations. *Proceedings of the National Academy of Sciences*, 117(3):1496–1503, 2020.

[75] Heng Yu, Liang Huang, Haitao Mi, and Kai Zhao. Max-violation perceptron and forced decoding for scalable mt training. In *Proceedings of EMNLP 2013*, 2013.

[76] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019.

[77] Zhongheng Zhang, Marcus W Beck, David A Winkler, Bin Huang, Wilbert Sibanda, Hemant Goyal, et al. Opening the black box of neural networks: methods for interpreting neural network models in clinical applications. *Annals of translational medicine*, 6(11), 2018.

[78] Kai Zhao and Liang Huang. Joint syntacto-discourse parsing and the syntacto-discourse treebank. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2117–2123, 2017.